Studies in Computer Science
and Artificial Intelligence

12

Problem Solving: Methods, Programming and Future Concepts

O.V. German
D.V. Ofitserov

North-Holland

PROBLEM SOLVING: METHODS, PROGRAMMING AND FUTURE CONCEPTS

STUDIES IN COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE 12

ELSEVIER

AMSTERDAM – LAUSANNE – NEW YORK – OXFORD – SHANNON – TOKYO

PROBLEM SOLVING: METHODS, PROGRAMMING AND FUTURE CONCEPTS

Oleg V. GERMAN

and

Dmitri V. OFITSEROV

Belarusian State University of
Informatics and Radioelectronics
Minsk, Republic of Belarus

1995
ELSEVIER

AMSTERDAM - LAUSANNE - NEW YORK - OXFORD - SHANNON - TOKYO

ELSEVIER SCIENCE B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE Amsterdam, The Netherlands

ISBN: 0 444 82226 7

No responsibility is assumed by the publisher for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions or ideas contained in the material herein.

This book is printed on acid-free paper.

Printed in The Netherlands

PREFACE

Problem solving is the very area of artificial intelligence (**AI**) which, probably, will never result in a complete set of formalized theories, in a kind of pragmatic philosophy, or in a "universal" applied discipline. Studying the questions concerning this area encompasses different concepts, models and theories.

In this connection the main accent falls on the theoretical framework (paradigm) of a problem solving system which should be considered from the viewpoint of the triad "human-problem-computer". In order to be effective, problem solving systems (or, more exactly, computer-aided problem solving systems (**CAPSS**)) should integrate human-solver's skill and abilities. That is, while humans are responsible for the "informal" part of problem solving activities the computer solves the "formal" part respectively.

When solving a problem, the human's properties are put to the forefront. It is the human's level of skill, role in the informal and creative aspects of problem solving, and the human's interpretation of the solution which affects the outcome most strongly. Humans capture, as we are often reminded , the " art of problem solving" while computers are ideal for carrying out extensive calculations and conducting search through the problem space.

Within this framework, both theoretical (mathematical) background and programming concept should be developed to provide solution for the following tasks:

- organizing the search activities of the human;
- automating the solving processes of the computer;
- creating program environment which provides an interface between the human and the computer;
- partitioning the tasks between the human and the computer in some optimal way.

These tasks are considered in the book. Our consideration is based on a new concept of **CAPSS** which incorporates the theory of weak methods, meta-procedures, and a programming paradigm which serves to support solving activities in **CAPSS**. (It should be noted here that some new terms, e.g. "weak methods", "meta-procedures", "**CAPSS**", etc., are explained in the following text. The reader can use the glossary for help at the end of the book.)

It is necessary to note that we consider weak methods mainly in the mathematical light. It is supposed, therefore, that the reader has a definite level of the mathematical culture (especially in the fields of discrete optimization theory and mathematical logic). However, the examples from these fields can be omitted without harm.

The book is oriented to the different groups of readers: mathematicians, specialists in computer sciences, and programmers. It can be useful for the post-graduates and the students, specializing in **AI** and applied mathematics.

The authors' contributions to this book are as follows: Dr. D.V. Ofitserov wrote Chapter 2 . The rest of the material belongs to Dr. O.V. German.

It is a pleasant duty to express our deep gratitude to the reviewers: Dr. Mitchell J. Nathan and Dr. David H. Green who performed a large work on improving the text. We are also obliged to senior editor Drs. A. Sevenster for his patience and interest in our project. We thank E. Germanovich for her invaluable help and assistance in preparing this book. Many warm words must be addressed to the students who helped us in our work. We are very thankful to Mr. Nigel Rix, Mr. Shaun Lynch and Mr. Andrew Horrall from Cambridge University (UK) for their help in the preparation of the book.

CONTENTS

This Page Intentionally Left Blank

INTRODUCTION

Problem solving is perhaps the oldest intellectual activity. Everybody faces various problems in his life. Very often, these problems are quite non-trivial. This book does not aim at covering every kind of problem. Rather, we will only deal with mathematical problems, or, more concretely, with a certain subset of such problems. However, we intend to display common approaches, principles and methods which can be applied to many different kinds of problems. We are interested in so-called weak methods and meta-procedures. Each method which does not warrant obtaining an exact solution or which is non-efficient with regard to its computational complexity is called *weak* in this book. Thus, any heuristic method is weak according to this definition.

By meta-procedure we understand a solving strategy (not a method) which points out how an exact solution to a problem (from a given class) should be found. The meta-procedures are characterized by the following paradigm:

(1) they are based on a heuristic search and (often) involve cutting mechanisms to reduce a search area;

(2) they extensively use logical inference to prove (or refute) hypotheses and assumptions which are typical of heuristic reasoning.

As we will consider a problem as a "black box", whose inner nature being known only partially, or even completely unknown to an investigator, these methods (and principles) are quite suitable for our conception of **CAPSS**. We shall call such problems partially-defined, having in mind their "black box"-properties. We proceed from the premise that if inner laws (i.e., the "nature") of a problem are known to the investigator, he can find the unknown values directly, and an exhaustive search may become substantially or entirely unnecessary. At the same time, as an exhaustive search is quite a rudimentary and labour-consuming method, other meta-procedures should be sought for. Expediency of a theory, which formalizes the methodology of meta-procedures (meta-methods, heuristics) and weak methods is based on three premises:

1) every new problem possesses some "black box"-properties until it is solved successfully;

2) there are problems with immanent property of partial definiteness (some examples are given below);

3) this theory models an approach to problem solving, which can form a basis for a so-called "intelligent problem solver" (e.g. computer program, robot, human).

Indeed, quite a lot of methods for solving partially-defined problems (i.e., problems regarded as "black box") are already developed, such as branch-and-bound, Monte-Carlo, (α-β) - procedure, dynamic programming, depth-first search, many iterative algorithms and most heuristic algorithms.

This study presents a certain class of meta-procedures and weak methods which are oriented to the problems with the following essential features:

1) considerable complexity of the original problem formalization in terms of finding an effectively computable procedure which connects the knowns to the unknowns;

2) need for relevant interpretation when solving particular problems;

These specific features manifest, for example, when we try to formalize rational strategies, other than simple exhaustive search, for finding an element with attribute α in a certain set R of elements. The following strategies are possible:

- using interrelations between objects;

- considering attribute β, so that

$\alpha \rightarrow \beta$ or $\beta \rightarrow \alpha$

where → - stands for a logical implication;

- finding consequences from α ;

- determining a characteristic function form over R, or an approximation to this function;

- determining a character of the set by means of random selection;

- decomposing R into subsets one of which includes the element sought for;

- introducing an additional element into R, which is obviously not included in the solution, and using the relations of this additional element with other elements of the set;

- changing the problem to a different interpretation;

- contracting the search space by means of the removal of obviously unsuitable elements;

- use of heuristics and expert estimations, etc.

To make clear the concept of a partially-defined problem, some examples of such problems are given below.

Example 1. The computation of the value of an integral which cannot be expressed as quadratures.

Example 2. Any NP-complete problem.

Example 3. Finding whole-number roots of Diophantes' equation.

Example 4. Finding roots of a polynomial with power equal to or greater than 5 on the basis of polynomial coefficients.

Example 5. A deducibility proof in first order logic.

Example 6. Finding syntax deduction in an arbitrary grammar.

Example 7. Problems of multicriteria optimization.

Of course, the examples given do not exhaust the whole list of similar problems. Using the examples we will try to reveal the essential features of the problems, which have been and still are of great interest to mathematicians and cognitive psychologists.

1. Absence of a universal method (in the case of algorithmically unsolvable problems), which results in solving every particular problem by means of low efficiency methods with no guarantee of a successful solution.

2. Impossibility to express, in functional and algorithmical form, the unknowns in terms of the knowns.

3. Infinite or extremely big tree of states (combinatorial explosion).

4. Incompleteness of the universe of discourse in terms of deduction of the necessary corollaries and theorems.

The foregoing considerations explain why we treat a partially-defined problem as a problem with inner regularities that are not completely known. Two ways of finding the solution can be proposed:

- disclosure of latent regularities;

- search for the solution without such disclosure.

Successful search for solution depends on three following components: existence of the solution in principle, the availability of knowledge sufficient to find a solution, availability of a procedure (algorithm, function) to extract the solution from the knowledge. Write E, K, R for these components respectively. The following table presents possible combinations, 0 and 1, denoting respectively the absence or existence of the corresponding knowledge.

	E	K	R
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

Cases 1 and 8 are trivial. Cases 2, 3, 4 are contradictory. Cases 5-7 remain which are interesting. According to the given classification ,a partially-defined problem proper corresponds to cases 5,6. However, having in mind to create an algorithm based on meta-procedures we are justified to include the traditional interpretation 7 as well. Case 6 is of great interest to educators and cognitive psychologists who study how problem solving with incomplete understanding is accomplished.

Thus, note as an intermediate conclusion, that a theory of solving partially-defined problems deals with the development of meta-methods for problem solving. At early stages of the solving they make it possible to find solutions of algorithmically unsolvable or NP-complete problems (with no guaranteed results, though). Consideration of such methods makes up the theoretical content of this book.

Our main goal consists of developing a theoretical framework which makes weak methods strong (i.e. enabling human-solver to find optimal and exact solutions for the partially-defined problems rather efficiently).

The second main feature of the book is its practical orientation to new types of program packages which implement a programming environment for a certain wide class of problems. This environment is essentially based on ideas of creative problem solving in the spirit of G.Polya. This new orientation includes two features:

(1) the package is function-oriented, i.e., the package provides powerful support for solving problems of a certain functional area, while programming language universality is retained enabling one to program any numerical procedure;

(2) the package includes an environment for problem solving, which does not require knowledge of the solving algorithm a priori and which is based on ideas of structured search for solutions in the spirit of G.Polya.

Historically, the main trends in languages and application packages (**AP**) are: improvement of programming techniques; increase of language's level and functional means; development of compilation theory; improvement in man-machine interface; programming automation.

When analyzing concepts of the development in languages and **AI**, the "human-problem-computer" triad should be considered. In our opinion, only (or mainly) the second part of the triad has been the focus of attention of users and builders of the programming languages. Pioneer studies of G.Polya, J.R.Slagle, A.Newell, R.Banerji, N.Nilsson, Wang Hao, V.Pushkin, V.Glushkow, S.Maslow, A.Tyugy and others deal with principles of problem solving by means of man and machine, and make up a basis for a new programming paradigm, oriented to problem solving.

A situation is quite possible when a human finds himself "face to face" with a problem. It is connected with the following factors:

- a human is likely not to be familiar with a problem and its methods of solution;

- the soiution supplied by a mathematician or taken from a reference book, is to be programmed, debugged and tested prior to its use;

- a suitable package may be unavailable, its delivery and installation may be time and money consuming; besides, the package may be not completely suitable, etc.

The following conclusion is suggested: the proper programming means should be integrated in **CAPSS** to provide necessary support in solving problems.

Fig.0.1 shows the structure of a new type of package. The package components are intended for the following purposes.

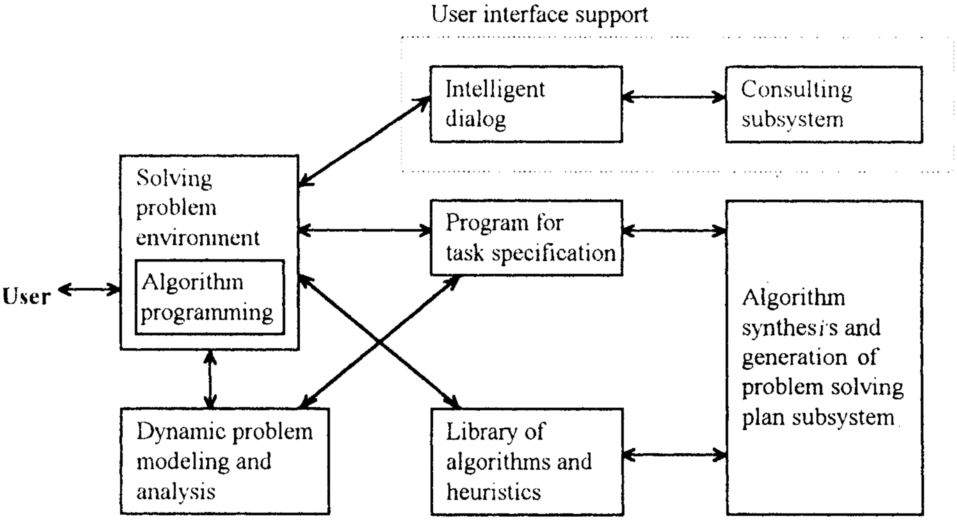

User interface support
Intelligent dialog
Consulting subsystem
Solving problem environment
Algorithm programming
User
Program for task specification
Algorithm synthesis and generation of problem solving plan subsystem
Dynamic problem modeling and analysis
Library of algorithms and heuristics

Fig 0. 1.

The problem solving environment is an integrating component which serves to control the program modules of the package; a special language is included to create and modify the computing context, intelligent dialog and environment for programming an algorithm. Its main purpose is the creation of a program for problem solving. A problem is considered to be stated if it is correctly specified in terms of syntax and semantics of an appropriate class of problems. The system supports creation of the specification for various categories of users, who are classified by the degree to which they know a problem's subject. It is supposed that the user realizes what he wants or that he can make it clear by means of a dialog with the system. The intelligent dialog subsystem and the consulting subsystem are responsible for user interface support.

The consulting system also implements the procedure of "introducing the problem". Such an introduction is aimed at refining a problem (what is to be found), initial data (what is known) and solving plan (how to solve the problem). The consulting system presupposes a close interaction with the algorithm synthesis and problem solving plan-generation subsystem.

The problem solving process is arranged in the following steps:

1 - preparation of problem specification (model);

2 - search for a solution;

3 - algorithmic programming;

4 - program execution using computer.

The search for a solving algorithm (step 3) reduces to the subset of the following problems:

- search for a suitable algorithm in the library;

- algorithm synthesis by specification according to the properties and relations of computability;

- algorithm synthesis by means of "extrapolation" (i.e., analogy) of appropriate heuristics over the problem specification;

- search for an algorithm performed by a human in the problem solving environment with the help of leading questions, prompts and meta-procedures (guided by the intelligent dialog subsystem).

Finally, the system simulation and dynamics analysis module provides for time representation of the solution obtained, e.g., simulation of a timetable or a time interaction.

It is necessary to make one last introductory digression.

This book describes approaches to problem solving in "human-machine" systems, which are human-oriented, where properties of a solver are put to the forefront. Really, if the problem solver is a human, the question of his (her) level of skill arises. Even a professional mathematician may be incompetent in the field of a given problem. Similarly, if the problem is solved by a computer, its possibilities are specified by methods available and by restrictions imposed by the well-known Gödel's theorem [1]. Thus, creation of a universal procedure for problem solving is algorithmically impossible. Our aim is different: we aim at making a machine responsible for the formal part, leaving the human to deal with the "informal" (interpretive) part. It seems reasonable to assume that, provided a problem (even a mathematical one) is transferred to the "interpretive plane" which is intelligible to a human-solver the man is able to find the solution in this interpretive plane. This is true even if he is not well-versed in the appropriate branch of mathematics. Naturally, it is difficult to give an example of such a plane for some mathematical fields, e.g., integro-differential equations. However, wide classes of problems may serve as a proving ground, such as discrete optimization problems on graphs and matrices. The system under consideration has to supply the user with an interface which allows him to treat a problem as an object (world) whose formal relations are "hidden" in its form but not in its essence.

The foregoing reasoning is certainly simplified, and user's level of skill (including logical abilities) should be taken into account, though we omit this issue for the present.

Primary studies in heuristic and game programming were mainly focused on problems of this kind. The specific character of these problems is closely related to psychology. Psychological aspects of the problem solving process are based on the heuristic concept, which in turn may be associated with an empirical regularity in the form of intuitive conjecture. This aspect of the search process may be called the "art of problem solving" (with certain reserve). As to its formal aspect, a theory is involved which is referred to as problem solving in English literature. The heart of the theory is a search through the problem space. The theory of problem solving is undoubtedly effective. However, there are some restrictions that cannot be overcome by this theory; in the first place, it concerns the contraction of the search area. Thus, progress in

[1] *Gödel K. Uber formal unentscheidbare Satze der Principia Mathematica und Verwandter Systeme I. - Monatshefte fur Mathematik und Physik, 38, 1931, S.173-198.*

problem solving requires extensive scientific research and methodological support of a meta-method theory including methods for search in a state space by means of a heuristic estimator, methods for restricting the search, methods for the synthesis of a chain of operators which convert objects into a different state, etc. Heuristic methods can be justified by means of the following pattern.

We can imagine different stages of the solving process:

a) a regularity exists and we (confident in this fact) are searching for it;

b) a regularity exists and we (not confident in its existence) are searching for it;

c) a regularity does not exist and we (confident in the opposite) are searching for it;

d) a regularity does not exist and we (not confident if it exists) are searching for it;

e) a regularity does not exist and we give up the problem supposing all efforts to be unavailing.

Analysis of alternatives a) - e) suggests the following conclusions.

Firstly, alternative (a) does not prove to be the only possible one, in contrast to the usual intuitive approach.

Secondly, problem solving is not only a search for a key regularity, but also a demonstration of its existence or absence. A note should be made that a human solving a problem, often tries to "kill two birds with one stone", namely: if a human finds a key regularity while solving a problem, the existence of this regularity is automatically proved. The opposite process (demonstration of existence of a regularity prior to search) seems to be more difficult for the following reasons:

- there may be no proof at all;
- demonstration may require greater effort than the search for a solution;
- proof of the existence of a solution does not leads directly to solving procedure.

Thus, at first sight, the case seems to be hopeless if a regularity does not exist while this fact cannot be proved. However, such reasoning is not relevant. Indeed, if there is no general regularity to find an efficient algorithm for solving general problem, it does not mean that it is impossible to find a particular solution using a heuristic approach, as often happens in practice.

Thus, the heuristic approach to problem solving is likely to be the only possible one, provided, that the nature of a problem is partially known or completely unknown to an investigator. On the other hand, many problems can be solved by the following patterns.

Pattern 1:

- some approximate solution is found using a heuristic algorithm;

- this approximate solution is used to contract the search area and to find the exact solution or to repeat the pattern from the beginning.

Pattern 2:

- some approximate solution is found using heuristic algorithm;

- if a criterion is known which is to be satisfied by the exact solution, its satisfiability is checked; if the criterion is not satisfied the pattern is repeated from the beginning, a new approximate solution being generated; if all possible (allowable) solutions can be found, this pattern guarantees the exact solution to be found.

Thus, another justification of heuristic methods is the fact that the using of these methods, guarantees exact solutions to be found.

Conception of the book

This book is an introduction to problem solving from the viewpoint of a theory of weak methods and a programming paradigm oriented to supporting problem solving.

The book investigates problem solving and is dedicated to a section of artificial intelligence theory, which interprets methods and postulates from different branches of mathematics. This is the main difficulty in the development of the appropriate theory.

When building systems for problem solving, the authors employ the approach based on the concept of an integrated program environment implemented in the **PROLOG** language. Such an environment includes three macrosystems oriented to the following groups of problems:

- equation forming (**ME**);
- theoretical multiple and logical problems (**SLE**);

- heuristic problems (**HP**).

Thus, this program environment covers a wide class of problems from polynomial equations to NP-complete problems of planning. The **ME** environment is based on a structural-functional representation of a problem tree, where the parent vertex is functionally determined through the child vertices. Thus, an equation corresponds to the recursive specifications, i.e. expressions of the following form

$$x = f(x).$$

The functional interpreter "reduces" a tree to its root vertex and transfers the "unreducible" expression to the input of the solver (for the time being, an algorithm is built to solve polynomial equations of power $n > 0$).

The **SLE** environment is based on methods for solving the following basic problems.

A set $S = \{s_i\}$ of objects and a set $\Psi = \{\psi_j\}$of relational equations which are defined on S, are given.

To find:

(a) arbitrary tuple S_L satisfying Ψ ;

(b) minimal length tuple S^* satisfying Ψ ;

(c) tuple S^F maximizing functional F.

Equations are called relational when they use operators of relations between objects (relational operators). The **SLE** uses the following relational operators:

#	-	incompatibility
$\rightarrow$	-	implication
$\multimap$	-	prohibition
$\succ$	-	precedence
$\rhd$	-	dominance
$\geq$	-	more/equal
:-	-	exclusive OR.

In terms of these relations, many problems of decision making can be specified as well as discrete optimization problems on graphs and sets. The general purpose

PROLOG-based solver is developed which allows one to obtain solutions in a reasonable time period for sets of 40 - 60 objects.

HP system is a procedure-oriented heuristic knowledge base for nonautomatic solutions of combinatorial and other optimization problems. The solving is built in the form of a search context tree. Each context is a set of relations and values for the unknowns, as well as calculational expressions. As a matter of fact, a user forms a context base dynamically; for this purpose he is supplied with the appropriate software for context manipulation (creation, deletion, storage, restoration, connection and dynamic unification). A user can use the heuristics library to solve many familiar NP-complete problems. To access a heuristic, the user has to specify the problem by filling up its frame. An example of problem frame:

<<type_medium>
<constraints_on_medium>
<constraints_on_solving_process>
<constraints_on_solution>
<structure_of_solution>
<properties_of_solution>
<relations_between_elements_of_solution>>.

A technological problem solving system (**TPSS**) is implemented as a prototype environment in MS-DOS with a modular structure and a total memory capacity of 750 K. The principles of **TPSS** are based on the concept of nonautomatic programming, which was earlier reported by the authors[2]. The **TPSS** supposes an active involvement of both a human and a machine in problem solving. This approach is different from the traditional philosophy of problem solving, where the computer's role is reduced to mere calculation, and from the automatic concept, where the man's role is reduced to problem specification.

[2] *O.V. German, E.I. Germanovich. O paradigme programmirovaniya,orientirovannoy na reshenie zadach. - Upravlyayuschie sistemy i mashiny. Kiev, 1993, No.1, pp.72-77 (In Russian).*

The theoretical content of the book includes the following issues: heuristic programming and application of heuristics to problem solving, calculus of relations and synthesis of solving procedures, theory of Φ-transformation as a formalization of meta-method for discrete optimization problems, implementation of the integrated environment for man-machine problem solving.

The history of the subject

The history of the subject goes back to the ancient Greeks who invented axiomatic method in mathematics and considered a proof as an essential part of it. They left some algorithmically unsolvable mathematical problems without answer(e.g., developing general algorithm for solving Diophantine equations).They introduced the notion of heuristic (studies by Archimedes and Pappus) and made other necessary premises for the further development of mathematics in the aspects of problem solving.

In "New Organon" F.Bacon (1561 - 1626) attempted to develop a method for invention, i.e., to develop a logic which allows study of the nature to be converted from casual activities into a systematic approach. Bacon considered the investigation of physical regularities rather than general mathematical problems. He listed methodologically significant aphorisms as meta-principles for problem solving on the one hand, and as empirical regularities in cognitive processes, on the other hand.

Bacon's aphorisms may be interpreted as follows:

(a) problem solving is not a random (blind) process; this process has to be directed by means of certain rules (heuristics);

(b) induction (generalization by analogy) is an elementary solving operator ;

(c) total study of a problem is possible only if the problem is considered in terms of a more general problem;

(d) the course of solving has to be obvious as well as composed of individual steps;

(e) regularities can be searched for in two ways - from general truths to particular ones, or vice versa; the latter way is essentially supported by examples and is preferred by Bacon;

(f) new regularities are revealed through old ones;

(g) human always inclines to assume more order than there is in reality (the presumption of regularity which is mentioned above);

(h) the greatest obstacle in the way of scientific progress and solving new problems undoubtedly proves to be ... human's despair and the assumption of the existence of the impossible;

(i) one should suppose that many regularities exist which have no parallel with already known regularities.

The patterns of problems given by Bacon are the following.

Single problems - these display regularities which are not found in other problems of this class.

Transitory problems - these reveal an intermediate character between representative and secretive problems (see below).

Representative problems - the regularity under consideration is displayed in the most evident way ("...because if every body takes forms of many natures in particular combination, then one form deadens, suppresses, develops and binds another form. Therefore, individual forms (laws) become obscure"[3]).

Secretive problems - these are opposed to representative problems ("...because they show the nature under study as if in embryo..."[4]).

Constructive problems - these employ regularities of one class.

Singular problems - these seem unusual as if isolated from other problems.

Deflective problems - the regularity seems broken (because of the influence by another, more powerful regularity).

Convoy and enmity problems - these are characterized by the presence or absence of certain regularity.

Extreme and limit problems - these show the limits for inner regularity,etc.

To find the factors which determine a certain inner regularity, Bacon offers a kind of calculus which in modern representation (especially with the reference to inductive logic by J.S. Mill) may be summarized as shown in Table 0.1.

[3] *quoted on F.Bacon. "New Organon", MSP, 1935, p.384, (in Russian).*

[4] *ibidem*

Table 0.1.

Presence of regularity and determination of nature-forming factors		
Regularity	Observable factors	Factor which induces regularity
L	a,b,c	
L	a,$\bar{b}$,c	
L	a,$\bar{b}$,$\bar{c}$	a
L	a,b,$\bar{c}$	
L	a,b,c	
L	a,f d	a (incomplete induction)
L	a,h,e	
L	a,b,c,d	
$\bar{L}$	$\bar{a}$,b,c,d	a
L	a∨b	
L	$\bar{a}$	b (deduction rule)
L	a,b,c	
L	a,d,e	
L	b,h,e	(a,b)
L	f,b,c	(hypothesis)
L	a	
L	$\bar{a}$	is unknown
L	a	a∨b∨x
L	b	x - unknown factor
L	g_1 (a,b,...)	all solutions of equation
L	g_2 (a,b,...)	$g_1(a,b,...)\&g_2(a,b,...)\&...\&g_n(a,b,...)\equiv 1$
. . .	. . .	
L	g_n (a,b,...)	
L_1L_2	a,b	
L_1	a,$\bar{b}$	
L_2	$\bar{a}$,b	b

Bacon's reasoning is notable for its maximal generality like that of his contemporaries and followers. His considerations are in the field of common logic, being valuable in terms of methodology rather than practical usage (nevertheless, the latter is by no means disclaimed). Anyway, Bacon shows the way based on experience. This point distinguishes him from, say, Descartes. The deduction of regularities from examples seems natural while specific deductive procedures depend on the type of regularity under consideration, and cannot be so general. Bacon's ideas are certainly of great value, for induction and incomplete induction constitute a basic mechanism for discovery in mathematics.

A substantial contribution to the subject was made by R.Descartes in his essay "Rules for a mind" written between 1619 and 1629. This study aims at, as Descartes says: "...directing a wit in such way that solid and true judgements are given about all subjects available" [5]. The purpose is achieved with the aid of a set ot rules which are methodologically similar to Bacon's aphorisms. Descartes adheres to the deductive method of cognition. As to a philosophical comparative analysis of deductive and inductive methods of cognition, it seems to be necessary and will be described below. Descartes' rules, like Bacon's, are of an extremely general, psychologically instructive nature. We give a brief account of these rules in terms of the subject under consideration.

(a) a problem should be considered from different positions (interpretations); e.g., physical analogs should be sought in the light of mathematical problems.

(b) a search must be developed from the simple to the complex, "... therefore, it is better to give up rather than study the complicated problems in which one is unable to differ the simple from the complex and is compelled to take the questionable for the trustworthy".

(c) acts of cognition must be evident. The evidence is assured in two ways: by means of intuition and by means of rules about deduction. By "intuition" Descartes means a state of mind "...so simple and clear that there is no doubt about we are thinking...". This thesis is of unquestionable significance. As a matter of fact, it essentially rests upon the philosophical solution for the problem of truth. The mathematical notion of truth runs across proof. In other words, something is considered true in mathematics, provided it can be proved. Meanwhile, there are two delicate sides of the issue.

Firstly, there exist truths which cannot be proved. Secondly, there is a problem about the validity of the proof itself. As to the philosophical criterion for validity based on experience, one cannot always apply it to mathematics. Indeed, how can one ascertain the validity of a statement which cannot be proved? The second side of the issue is

[5] *R.Descartes. Pravila dla ruckovodstva uma - Moscow.:SSEP, 1936, P.174,(in Russian).*

important as well. The validity of a proof appeals to common sense and maybe to the knowledge of a limited number of experts. Any proof can be divided into elementary steps of premises and conclusions. The elementariness means that it is inexpedient to look for more elementary formalisms. As to the validity of each elementary step, it is learned through evidence, or as Descartes says, through intuition.

(d) a search must be purposeful (not trial-and-error). It supposes availability of a special method in the form of a set of such simple and precise rules that "...following these rules, one is always prevented from taking false for true...and is obtaining knowledge which makes it possible to learn everything within human reach".

Thus, Descartes postulates that a method to learn truth exists in the form of a set of rules, even if he does not mention directly the existence of such a method, or its universality. Subsequently, a universal system of rules was sought for by G. Leibnitz. However, K. Gödel destroyed this delusion. He showed that there are truths in mathematics that cannot be proved. As a corollary, a universal system is impossible which could search for such truths and prove them. This problem can be examined from the different viewpoints, though.

Firstly, the variety of the world is infinite, which means that the set of truths within the reach of a man, will always be incomplete. Secondly, availability of a problem solving system, even if non-universal (but still effective), is important in itself, from both theoretical and practical viewpoints. The value of an unprovable truth is determined by its practical helpfulness.

(e) search for a truth is based on such order as to deduce the unknown from the known.

(f) when solving a problem, all the factors involved must be covered.

(g) a problem must be simplified to such a degree that its inner nature still remains valid.

(h) a proof must be thorough so that each of the steps is evident and is not overlooked.

The comparison of Descartes' rules and Bacon's aphorisms offers a number of common points. The principal difference is the choice of the method (Descartes adheres to deduction while Bacon prefers induction). What is the philosophical difference between the methods?

The deductive method establishes truths which follow from premises according to formal rules. On that ground, this method may be called "closed"; in other words, if there exists a set of true expressions E and a set of rules R, then deducibility of true expressions from E by means of R can be written as follows

$$\begin{aligned}
&\varphi_0 = R(E)\\
&\varphi_1 = R(E \cup \varphi_0)\\
&\varphi_2 = R(E \cup \varphi_0 \cup \varphi_1)\\
&\quad \ldots\\
&\varphi_k = R(E \cup \varphi_0 \cup \ldots \cup \varphi_{k-1}).
\end{aligned}$$

The subscript of φ_i refers to deduction length i. It is supposed that each true expression can be deduced, the length of deduction being finite. In other words, all true expressions are assumed to be finitely deducible. However, a deduction with length of 10^{20} is obviously unattainable. This is the first constraint imposed on the deduction method. Besides, due to Gödel's theorem, there exist truths which can be expressed in formal arithmetical systems and which cannot be proved within the systems. Such truths cannot be obtained by the deduction method. This constraint of deduction turns into an advantage in the case of induction, which reveals truths "beyond" certain formal systems, even if these "induced" truths cannot be proved within the system. Thus, the induction method "expands" the boundaries of formal systems. This feature is important for problem solving.

Leibnitz (1646 - 1716) is to be mentioned next in this list. Leibnitz highlights principles of cognition, i.e."...all basic truths which suffice to obtain all necessary conclusions"[6], as well as art of employing these principles, namely:

- art of reasoning;
- art of discovery;
- art of use.

[6] *quoted on G.V. Leibnitz. Works in 4 volumes, Moscow: "Thought", 1982, (in Russian).*

Leibnitz defines each of these arts through maxims. The art of reasoning is expressed by the following maxims.

a1) only an evident statement is recognized as truth, as well as all its evident corollaries;

(a2) otherwise, one has to content oneself with the probability of authenticity...any conclusion drawn from such truth being even less authentic;

(a3) to derive one truth from another, some inseparable link has to be held. The art of discovery implies the following.

(b1) to get to know some object, all its properties have to be examined;

(b2) a complex object requires the investigation of its individual parts (analysis);

(b3) the analysis must be complete ;

(b4) an object should be considered as a whole;

(b5) investigation should always be started from the easiest points, i.e. the most general and simple points.

Here is an insight which deserves special attention from the viewpoint of our subject: "...by universal science I mean science which teaches to discover and prove every knowledge on the basis of sufficient data...When studying any science, one should try to find principles for discovery. These principles are related to some superior science, in other words - art of discovery; these principles may prove to be sufficient to derive all the other truths, or at least the most useful truths, without burdening oneself with too numerous rules...I aim at disclosing of problem solving methods rather than solutions to individual problems, for a single one method includes an infinite number of solutions".

This idea of Leibnitz proved to be utopian; nevertheless, the new scientifical lead has been established. This idea is practical enough, provided the surplus generality is removed.

The art of problem solving is still substantially based on psychology and intuition. As H. Poincare has noted "the rules in force are extremely delicate and subtle; they are hardly expressible by exact words; they are more easy felt than defined..." and further:"...the fact is more than evident that the emotional component is essential in discovery or invention, this fact is confirmed by many thinkers...".

We do not have for an object to describe psychological results in the field of problem solving, such as trial-and-error theory, Gestalt psychology, heuristic programming (in its psychological aspect). In this connection, fundamental studies by G. Polya should be mentioned. Acquaintance with these works is advisable. One should specially refer to studies by Soviet authors V. Pushkin, D. Pospelov, S. Maslov, Shanin et al.

Here we draw the main conclusions. Historically, approaches to problem solving have been based on the concept of purposeful search using a certain set of rules and strategies. The attempt to create some universal calculus based upon this concept was unsuccessful. Moreover, the presence of a "psychological aspect" in problem solving prevents the rules from simple formalization. Thus, two main approaches are possible:

(a) development of principles for such systems that solve problems with the participation of a human, where the degree of human's involvement allowing a search for a solution must be formalized;

(b) development of such systems where human's part is decisive (i.e., a human possesses properties of a problem solver).

These approaches will be examined below, their "theoretical" development suggests the following directions:

- creation of "particular" calculus (calculi) of problems based on mathematical logic tools, on methods for search in a state space and organized exhaustive search, etc.;
- creation of theory of meta-methods for problem solving from the semiformal position of their application, as well as principles of problem solving in "human-computer" system, where the human-solver remains an "informal interpreter" of the solving process.

State of the art

Consider the basic concepts for solution search by means of "extraction" of the solution from the proof of computability theorem [1, 2]. Interpret predicate $P(X_1,X_2,\dots,X_k,Y_1,Y_2,\dots,Y_p)$ as procedure P which calculates output variables $Y_1,\dots,Y_p$ from input data $X_1,X_2,\dots,X_k$. Any description of the universe of discourse involves the specification of the following components:

- information about objects and interrelations in the universe of discourse;

- rules which are specified by predicates P(...) and which determine computability relations on a set of objects in the universe of discourse. Suppose $S_1,...,S_k$ are sets. Subset R of the Cartesian product $S_1*...*S_k$ is a relation on $S_1,...,S_k$.

Tuple of $< l_1,...,l_k>$ satisfies R if $l_1 \in S_1$, $l_2 \in S_2,...,l_k \in S_k$, $< l_1,...,l_k> \in R$.

Suppose $\mu_1,...,\mu_k$ are definitional domains for variables $X_1,X_2,...,X_k$. Definition domain for relation R $(X_1,X_2 ,...,X_k)$ is Cartesian product $\mu_1*...*\mu_k$.

Functional relation

$$F_v:(X_{v1},\ldots,X_{vk})\mapsto(X_{vk+1},\ldots,X_{vs})$$

may be interpreted as a procedure or as predicate

$$P(X_{v1},X_{v2},\ldots;X_{vk+1},\ldots,X_{vs}).$$

Axioms of the universe of discourse are defined by a collection of relations and functional bindings:

$$\sigma=< R_1,\ldots,R_n;F_1,\ldots,F_m >.$$

The problem interpreted in such a way, can be represented as follows

$$X_{inp}\mapsto Y_{out}$$

where X_{inp} - input data, Y_{out} - data to be found. This dependence has to be derived from axioms of the universe of discourse:

$$\sigma\vdash F_x,$$

$$F_x: X_{inp}\mapsto Y_{out},$$

where $\vdash$ - deducibility relation.

If $\sigma\vdash F_x$, either σ contains F_x , or F_x can be found through the sequential application of rules for deduction of σ. As the deduction rules, Armstrong's axiom patterns of database relational algebra can be employed:

1. Reflexivity:

$X \mapsto X.$

2. Replenishment:

$X \mapsto Y$ involves $XZ \mapsto Y.$

3. Additivity:

$X \mapsto Y$ and $X \mapsto Z$ involves $X \mapsto YZ$.

4. Projectivity:

$X \mapsto YZ$ involves$X \mapsto Y.$

5. Transitivity:

$X \mapsto Y$, $Y \mapsto Z$ involves $X \mapsto Z$.

Example. Consider a problem with axioms σ of the form:

$a\alpha \mapsto h$
$b\beta \mapsto h$
$hc \mapsto S$
$h\alpha b \mapsto c$
$h\beta a \mapsto c$
$abc \mapsto S$
$h\alpha \mapsto a$
$h\beta \mapsto b$
$Sab \mapsto \alpha$

which is defined on a triangle shown in Fig. 0.2. Let us try to prove the deducibility:

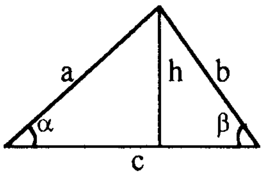


Fig. 0. 2.

$\sigma \vdash (h\alpha b \mapsto S)$,

In this case, the deduction is performed through the following sequential steps:

1. $h, \alpha, b \mapsto c$ /* assumed axiom */
2. $h, \alpha \mapsto a$ /* assumed axiom */
3. $h, \alpha, b \mapsto a, b, c$ /* axiom of additivity and reflexivity */
4. $a, b, c \mapsto S$ /* assumed axiom */.

If each deduction rule used, is assigned to a certain program, then the chain of the deduction rules will be interpreted as a linear program, which calculates the wanted output values through the given input.

However, such a formalization does not allow one to write directly the recursive dependences; for example, operator

$A = A - B$

would take form of

$A, B \mapsto A$,

which, in terms of Armstrong's axioms, considers "A" as the same object both from the right and from the left of "$\mapsto$". The operators of this sort require a special representation by means of auxiliary objects. To extend resources for structure synthesis of programs, predicates $Q, Q_1, Q_2, \ldots$, are introduced into the language; the predicates are implemented through programs $q, q_1, q_2, \ldots$, which calculate logical values of these predicates. Introduce the following expressions

$$Q \to A \underset{f}{\mapsto} B$$

which are interpreted in the following way: "if Q is true, B can be calculated from A by means of procedure f". The following deduction rule is added to the language to synthesize branching programs

$$\frac{Q_1 \vee Q_2 \vee \ldots \vee Q_n; Q_1 \to A \underset{f}{\mapsto} B; Q_2 \mapsto A \underset{f}{\mapsto} B; \ldots; Q_n \to A \underset{f}{\mapsto} B}{A \mapsto B}$$

where at least one of predicates $Q_1, Q_2, \ldots, Q_n$ is always true. Paper [2] has shown that branching programs can be built by means of this rule. The indexing of variables has been used in [1] to represent recursive rules and cycles. For example, the following rules are admissible:

$$A[I] \underset{f}{\mapsto} A[I+1]$$

$$A[I] \underset{g}{\mapsto} B[I]$$

where I is an index.

Thus, operator A=A - B is associated with a functional dependence of the following form:

$$A[I], B[I] \mapsto A[I+1].$$

Examine this approach to the concept of problem solving. It is based on a priori knowledge of computability relations over the set of objects. Rigorously, two sets of such relations can be mentioned:

main relations - relations between main objects in the universe of discourse (main object exists in the universe of discourse);

auxiliary relations - relations written with the use of auxiliary objects which have no interpretation in the universe of discourse. E.g., in expression $A[I] \to A[I+1]$, I is such an object. Labels and conditionals can be classified among auxiliary objects as well.

The introduction of auxiliary objects is associated with the extension of problem's axiomatics. As this extension is not formalized, there is no regular approach to its implementation. By way of example, consider the problem of finding the greatest common divisor (GCD) for two integer numbers A and B. The solving algorithm can be written as follows:

```
M:if $A > B$, then assign $A = A - B$
otherwise
if $A < B$, then assign $B = B - A$
otherwise, terminate program with answer: $GCD = A = B$.
repeat from mark M.
```

In this case, using auxiliary relations, a human-solver writes Euclid's procedure itself rather than the property upon which it is based. Lately, an approach to solution synthesis is widely spread employing the specification of the problem [3 - 6]. In case of the problem of GCD, the specification could be written as follows:

```
PROBLEM (integer (A,B,C,X,Y))
GIVEN: A,B;
TO BE FOUND: C;
PROPERTIES:
(A:C=X)&
(B:C=Y)&
(C→max)
```

$$\begin{bmatrix} ((A{:}C = X)\& & \\ (B{:}C = Y)\& & \\ (A <> B) \rightarrow (|A - B|{:}C = |X - Y|)) & \\ (C{:}C = 1) & \end{bmatrix}$$

The specification of the same problem in PROLOG notation [5,7] is as follows

```
GCD(C,C,C).
GCD(A,B,C):-
A <> B,
abs (A,B,D,Z),
!,
GCD(Z,D,C).
abs(A,B,D,Z):-
A > B,
Z = A - B,
D = B.
abs(A,B,D,Z):-
A < B,
Z = B - A,
D = A.
```

The key rule for GCD(X,Y,Z) says that Z is the GCD for X and Y. In the first case, if X = Y, then Z = X = Y = C, i.e. the rule takes form: GCD(C,C,C). If X = Y, then GCD(X,Y;Z) is recursively determined through GCD(X - Y, min(X,Y),Z); D denoting min(X,Y).

This approach to problem solving supposes the man-made specification to be complete and consistent. It is clear that removing the square-bracketed part of the given specification, we have a partially-defined problem which can be solved with the aid of a human-solver. PROLOG uses a procedure for solution search through specification, which is a depth-first method based on a purposeful search for alternative solutions. This approach as well as the previous one is limited in terms of the use of computability relations. The limitation results from the requirements to describe a given problem completely and consistently. To meet these requirements the user must be a master of the subject which is not the normal case.

Another trend in problem solving theory is associated with search for routes in a state space. Every object $I_i(i = \overline{1,R})$of a problem is associated with a set of characteristics (attributes, properties, functions etc.) P_I^t = $<p_{i1},p_{i2},\ldots,p_{ii}>$, which determine element l_i at instant t; index t indicates that the values of attributes p_{ik}, $k = \overline{1,i}$ are defined for this instant).

The designator-state σ^t at instant t is a collection of sets $P_i^t\,(i=\overline{1,R})$ which correspond to objects of the system.

Define a basic operation ⊕ of an attribute setting for the description of intra- and inter-element interaction in terms of theoretical multiple operations of union U and difference \. The co-existence of mutually exclusive attributes p and $\bar{p}$ in the same set should be considered inadmissible, as well as the co-existence of attributes of the same nature showing different values. Suppose that values of any attribute fall within certain discrete domain, while at any instant the attribute is specified by only one value from this domain. Include null value in the range, bearing in mind the nature of the introduced operation ⊕, which is formally described in Table 0.2.

Table 0.2.

Definition of setting attribute oneration ⊕

Values of attributes before the operation	The defined set of attributes	Resulting set
$P=<p_1=\alpha_1,p_2=\alpha_2,\dots,p_n=\alpha_n>$	$\oplus<\varnothing>$	without modification
$P=<p_1=\alpha_1,p_2=\alpha_2,\dots,p_n=\alpha_n>$	$\oplus<p_1=\beta_1>$	$P=<p_1=\beta_1,p_2=\alpha_2,\dots,p_n=\alpha_n>$
$P=<p_1=\alpha_1,p_2=\alpha_2,\dots,p_n=\alpha_n>$	$\oplus<p_1=\varnothing>$	$P=<p_2=\alpha_2,p_3=\alpha_3,\dots,p_n=\alpha_n>$
$P=<\varnothing>$	$\oplus<p_1=\alpha_1,p_2=\alpha_{2>}$	$P=<p_1=\alpha_1,p_2=\alpha_2>$
$P=<p_2=\alpha_2,\dots,p_n=\alpha_n>$	$\oplus<p_1=\alpha_1>$	$P=<p_1=\alpha_1,p_2=\alpha_2,\dots,p_n=\alpha_n>$
$P=<p_1=\alpha_1>$	$\oplus<p_1=\alpha_1>$	$P=<p_1=\alpha_1>$

Definition. Interaction patterns between objects A and B, which define the operation ⊕ on sets of the objects A and B, are specified through a collection of predicate formulae according to the following expressions

$$F_i^{A,B} \rightarrow \oplus < \rho_A, \rho_B > \qquad \textbf{(0.1)}$$

where $\rho_A(\rho_B)$ - collection of attributes being specified in sets P_A^t (P_B^t); $F_i^{A,B}$ - predicate formula found by induction:

1) an expression derived from a predicate symbol through replacing its variables with other variables (not necessarily different ones), is a formula;

2) if X and Y are formulae and u is a variable, then expressions $\neg(X)$, $(X)\&(Y)$, $(X)\vee(Y)$, $(X)\rightarrow(Y)$, $(X)\leftrightarrow(Y), \forall u(X), \exists u(X)$ - are formulae as well.

The operation $\oplus$ mentioned in (0.1), "works" if the value of the corresponding predicate formula is true.

Evidently, a set of formulae like (0.1) formally defines the logical structure F of a problem and its quantitative aspect through the attributes of interacting objects of different nature: space-time attributes, state-quality attributes, quantitative attributes etc. The problem of our interest is like this: for given triad $< F,\sigma_0,\sigma_k>$ where σ_0,σ_k are two different designators, an interaction tuple $F^*= <F_{i1},F_{i2},...,F_{iz}>$ has to be found so that the sequential execution of interactions in F^*, beginning from σ_0 and F_{i1}, converts the system into designator-state σ_k. This problem will be examined in the book. The theory for solving problems of this kind, employs methods of search in a state graph using heuristic estimator [8-10].

Give an example. Consider a board with six squares and four counters numbered 1, 2, 3, 4 (Fig.0.3a). A counter may be only moved to adjacent vacant square. A movement sequence is to be found so that all the counters are placed as shown in Fig.0.3b. Design the state graph G for this problem. Label the states by $S_0,S_1,..,S_k$, where S_0 and S_k correspond to Fig.0.3a and Fig.0.3b respectively. Two vertices-states S_m and S_n are connected with an arc provided a counter can be moved from S_m to S_n in one movement. Display only that arcs which do not form cycles within G. A part of the graph G is shown in Fig.0.4. One can see that vertices have several output arcs which give alternative sequels in solving. The concept of search for solution by means of heuristic estimator is like this.

The following value is found for each vertex v

$$f(v) = g(v) + h(v)$$

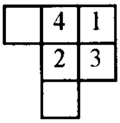


Fig. 0. 3. a)

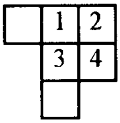


Fig. 0. 3. b)

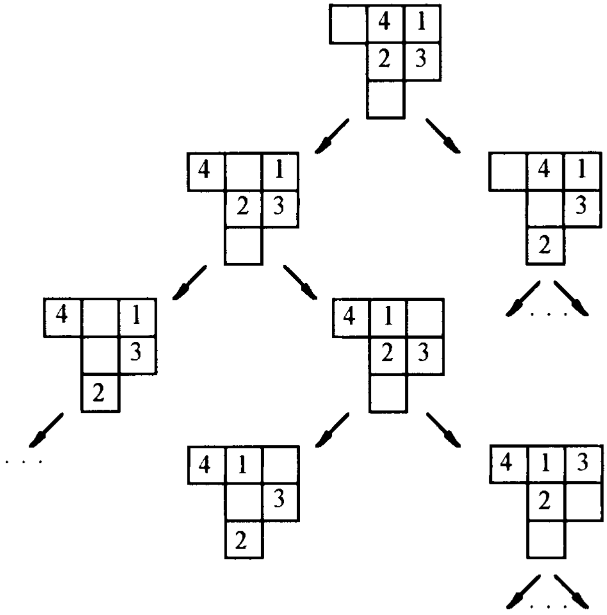


Fig. 0. 4.

where g(v) - route length from S_0 to S_v (S_v is the state corresponding to the vertex v;

h(v) - estimated route length from S_v to S_k;

f(v) - resultant estimator.

Suppose $v_1, v_2, \dots, v_z$ - are candidates for a sequel (alternative movements of the counter in the current state). Then the vertex with minimal value of f(v) is to be chosen.

Give a search algorithm using the estimator f as described in [5]. Suppose S^o is a set of vertices already chosen, S is a set of vertices which are candidates for sequel, $S \cap S^o = \varnothing$, v_0 is the initial vertex, v_k is the terminal vertex in G; let G(v) denote child vertices of v, the vertices already passed being eliminated.

Step 1. Place v_0 in S and calculate $f(v_0)$.

Step 2. If S=∅, then fail. Otherwise, go to step 3.

Step 3. Choose vertex v_x from S so that $f(v_x) = \min\limits_{V_y \in S} f(V_y)$ and place it in S^o on removal from S .

Step 4. If $v_x = v_k$ then end (the route is found). Otherwise; go to step 5.

Step 5. Find $G(v_x)$. If $G(v_x) = \varnothing$, go to step 2. Otherwise, find $f(v_i)$ for each vertex $v_i \in G(v_x)$.

Step 6. For each v_i:

if $v_i \notin S^o \cap S$, then place v_i in S;

if $v_i \in S \cap G(v_x)$, then assign to v_i the least of its estimations $f(v_i)$;

if $v_i \in S^o \cap G(v_x)$, then remove v_i from S^o and place

it in S with the least estimation $f(v_i)$;

in other cases, do not change S and S^o.

Step 7. Go to step 2.

The main difficulty in using this algorithm is to find h(v) to provide effective contraction of the search area. In this connection, the following points should be taken into account.

Firstly, it is not always necessary to search for the shortest route in G, for any finite solution can be admissible.

Secondly, there is no indication about the form of h(v).

Thirdly, the practical efficiency of this method is questionable in case of large graphs G.

To decrease the graph size (i.e. to contract the search area), [6] suggests splitting a problem into particular ones. For example, the first particular problem is to transfer counter 1 to its destination square, the location of other counters being insignificant. The second particular problem is to transfer counter 2 to its final position, the solution to subproblem 1 being kept. Subproblem 3 is to transfer counter 3 to its destination, the final location of counters 2 and 1 being kept, etc. Detailed examination of this concept is omitted here, as well as possible modifications of search algorithm for state graph.

The common point of these modifications is a regularization of the search procedure which ensures the result.

All the approaches described above have got some support in certain programming environments. Among the most popular systems the following ones may be listed: PROLOG, Lisp, Planer, PRIZ, ABSTRIPS, SITPLAN and others.

Thus, the development of problem solving theory is urgent both theoretically and practically in the following aspects:

- developing an approach based on using weak methods and meta-principles (meta-procedures);

- creating a programming environment oriented **to** supporting problem solving activities of a human-solver.

This Page Intentionally Left Blank

Chapter 0

PROBLEM CLASSIFICATION. INTRODUCTION TO THE SOLVING METHODS

0.1. What is a problem?

When speaking of a problem one uses such words as "to find", "solution", "unknown variable (s)", "search", "model", "problem states", "goal", etc. We admit that these terms are primary, i.e. they cannot be formulated by means of some other simpler notions. It may be supposed that the content of a primary notion is intuitively clear and unequivocal. Therefore when one is asked what it means "to find a solution to a given task?" it is automatically supposed that every other person would interpret this question in the same way. However, the opposite question " is a given solution really a valid one to the problem?" may sometimes be too difficult to answer (or even impossible to answer).

Definition of terms. Let us introduce the terms directly connected with the main subject of the book. It is quite natural from the consideration above that every "definition" of a primary notion is not a definition in the sense of formal logic. There are two goals of such definitions:

(i) to outline the conceptual field of a discussion;

(ii) to set a primary basis (notation) for further development of a subject under consideration.

A variable is an object which takes a meaning from a given domain. Sometimes the following notions are thought to be equivalent, i.e.

<variable> = <object> = <unknown-value>.

The distinction is provided due to the existence of an object and its denotation (or an object and its abstract essence).

A variable may be bound or unbound. In the first case one may consider the variable to have a value. In the second case the variable is not assigned a value. It is quite natural to consider the very notion of a value as another object of the definite type. From this viewpoint a bound variable is a pair <V_1 ,V_2> with an original object V_1 and derivative object V_2 standing for V_1 . We do not point out the nature of the derivative object and only require for V_2 to be a concrete object from some set of objects. Thus, the denotation

$$< X, \text{"} john \text{"} >$$

may easily be interpreted as assertion that the variable X is bound by a symbolic constant "john", or a symbolic constant "john" stands for the variable X. Some difficulties arise when V_2 represents a so-called self-definition (i.e., recursive definition), for instance,

$$< X, X^2 - X + 1 >$$

or in more convenient form

$$X = X^2 - X + 1.$$

If the last is interpreted as an equation, then one may deduce that X = 1.

If the last is interpreted as a substitution for X then to avoid an ambiguity one needs to use the representation

$$< X, \text{"} X^2 - X + 1 \text{"} >$$

or

$$< X, Y^2 - Y + 1 >$$

if replacement is valid.

From the above considerations one may deduce that a pair <X, Z> with unbound variable Z defines an unbound state of variable X if Z is not a recursive function of the same single variable X.

Another interesting case is a partly-bound variable X, i.e. with representation

$$< X, \text{ ["}john\text{", "}bill\text{", "}dick\text{"]}^{*} >$$

where X takes a meaning from a given list of symbolic objects, i.e.

$$X = \text{"}john\text{"} \; \dot{V} X = \text{"}bill\text{"} \; \dot{V} X = \text{"}dic\text{"}$$

where $\dot{V}$ is exclusive "OR"- operation.

We used an asterisk to denote the fact that the list above does not stand as a whole for the variable X.

The case under consideration occupies an intermediate state between the previous two.

A constant object is represented by <C, C>, where C is a concrete object, i.e. one may say that a constant object represents itself. The designation <V, V> with the bound variable V is of the same kind. To eliminate excessive identifiers a constant object is designated as a single one, i.e. instead of <C, C> it is simply written as C. Variables and constant objects will be called terms.

To sum up all the above considerations of the object we have to introduce the notion of a problem state. A problem state is a set of pairs $<V_{i1}, V_{i1}>$, $i = \overline{1, I}$ defined for all the terms we deal with in a problem.

All the objects in a problem form a system of different relations. From the mathematical viewpoint, the n-ary relation $R(t_1, t_2,...,t_n)$ is defined as a subset of the Cartesian product

$$T_1 \; * \; T_2 \; ... \; * \; T_n$$

where T_i is a definition space for term t_i.

An example of a 1-ary relation may be the following

prm_rel (X)

where prm_rel is the symbol for the relation "to be a prime number" and X stands for an arbitrary integer number. Thus, the relation

prm_rel (12)

is false since 12 is not a prime number, and

prm_rel (13)

is true.

An example of 3-ary relation may be the following

rect_sq (X,Y,S)

where rect_sq is the symbol of relation "X (a width) and Y (a height) are the sizes of a rectangle and S is its area, i.e.

S = X * Y.

Thus,

rect_sq (5,2,10) is an example of true relation
and rect_sq (5,3,10) is an example of false one.

Besides relationships of the above type there are functional equations (functions) as in the example below

$y = x^2 + 2x + 1$
or $y = \int e^{-x} dx$.

One may consider these equations as the particular cases of the equality relation (=), i.e.

$=(y,\ x^2\ +2x+1)$
and $= (y, \int e^{-x} dx)$.

Let us consider a function

$$y\ = x^2\ +2x\ +1.$$

One may simply test that the state S' = (<y,4>, <x,2>) does not suit it, whilst on the contrary the state S" = (<y,1>, <x,0>) does. The notion of a <u>solution</u> is therefore frequently associated with the unknown state S suiting all given relations. The solution understood in such a manner will be called an <u>interpretation</u> for a given <u>model</u>, i.e. the set of original relations. To extend the notion of a solution one may need to find such an interpretation which satisfies additional criteria, for instance, such as

$$\text{maximize}(4x_1 - 3\overline{x}_2 + 5x_1x_2 - 2\overline{x}_1x_3 + 3\overline{x}_1)$$

where x_1 ,x_2 ,x_3 are boolean variables.

In this context "to find an interpretation" means to build a solving procedure which warrants obtaining an optimum interpretation. All of the above give us a possibility to denote problem P by the 6-tuple

P = <Model, Initial_State, [Criterion,] Solution, Solving_Procedure[,Proof]> **(0.2)**

The members in square brackets are optional and may be omitted. It is worth noting that the "proof" is necessary as a certification of the solution validity (let us recall the question "is a given solution really valid?" formulated earlier).

Now we may go directly to problem classification.

0.2. Problem classification

Considering definition (0.2) one may establish 64 different configurations of a problem P by the assumption that every member may take only two possible values: {0,1}, where 1 means "it is known" and 0 - "it is not known". The typical combination is <1,1,1,0,0,0> and two absurd combinations are <0,0,0,0,0,0> and <0,0,0,0,0,1>. It is also beyond our interest to explore the trivial case <1,1,[x],0,1,1>, x $\in$ {0,1}.

The problems we shall discuss form the following groups:

(i) finding suitable interpretation (i.e., variable values) for the model given in the following problem specification

P = <Model, Initial_State, Solution, Solving_Procedure, [Proof]>.

Here, Solution represents such an interpretation.

(ii) finding optimum interpretation for the model given in the full specification (0-2) which includes Criterion(-ia).

(iii) finding solving-procedure.

Indeed, the first two groups (i)&(ii) are provided with a powerful and tremendous library of solving methods (**LSM**). However, there are three important problems concerning these groups which are connected with further investigations in the first instance:

(P1) the development of "universal" solving procedures and meta-methods to constrain uncontrollable growth of **LSM**. The idea of a universal solving program was strongly criticized but the situation has drastically changed since the appearance of the logical programming languages, and their wide utilization starting from the early 70's. It is correct to speak of not one but a variety of "universal" methods to satisfy the needs of creative mathematics. The marginal line between **LSM** and "weak" methods separates two large classes of problems

P = <*,*,*,*.0,0>

and P = <*,*,*,*,1,1>

where "*" is either 0 or 1.

(P2) there is a special class of problems (so-called **NP**-complete problems) which are still solved by the inefficient methods and there are no warrants to find efficient algorithms to solve them.

(P3) it is often difficult to divide an original problem into a set of subproblems P_1, P_2,...,P_0 with concerted inputs and outputs.

And, finally, the last group (iii) is primely connected with such "universal" weak methods.

Let us proceed from the problems comprising the group "Finding Suitable Interpretation".

0.3. An approach to building an interpretation calculus

This section deals with the logical inference system on the basis of interpretation calculus developing Beth's ideas and rules of paramodulation for equality calculus. It is assumed that the reader is acquainted with the predicate calculus (see, for example, [7,8,9]).

0.3.1. The theorem of solution existence

The idea of using theorem proving techniques for solving problems was proposed by R. Kowalsky [8]. This idea consists in proving the following theorem [1] of solution existence:

$$\forall x \exists y\ R\ (x, y) \tag{0.3}$$

asserting that for every input set x of the problem there exists a solution y obtained by some solving procedure R.

If a problem is written in a suitable formal language then to obtain a solving procedure R one needs to realize the following scheme:

1) to formulate a theorem of solution existence;

2) to build a proof of this theorem;

3) to extract a solving procedure R from the proof.

The scheme above is connected with two main problems:

(i) to prove that a solution really exists;

(ii) to find a solution.

We consider both these problems below.

0.3.2. Preliminary remarks

Beth [10] proposed a procedure to prove (or to refute) deducibility of a formula ψ from the set of formulae F on the basis of finding interpretation I satisfying every formula from F and refuting formula $\overline{\psi}$ (not-ψ). Using this common principle we suggest for every formula of a given formal task representation the method for building individual interpretations in the form of a disjunction

$$(I_{i1} \vee I_{i2} \vee \ldots \vee I_{iz}),$$

where every individual interpretation I_{im}, m = $\overline{1,z}$ provides consistency of a given formula. Note that the individual interpretations I_{im} , m = $\overline{1,z}$ do not form the whole Herbrand universe but correspond merely to its finite part. Besides, every individual interpretation provides consistency of a given formula what is not equivalent in general to its truth. To clear this fact let us consider the formula

$$\overline{\exists} x\, P(x)$$

($^{-}$ is a symbol of negation).

This formula is consistent with two interpretations:

I_p : (P1 = □) and $I_{\overline{p}}$: (P1 = ALL)

where

□ stands for an empty interpretation (the null set);

I_p is an interpretation for formula P(.);

$I_{\overline{p}}$ is an interpretation for formula $\overline{\mathrm{P}}$(.);

"ALL" corresponds to every conceivable value from a given domain;

P1 is the designation for an argument of predicate P(.) or $\overline{\mathrm{P}}$(.).

Suppose an original problem is formalized by a set of formulae

$$F = \{\varphi_1, \varphi_2, \ldots, \varphi_m\}$$

with interpretations $I_{\varphi 1}$, $I_{\varphi 2}$,..., $I_{\varphi\mu}$ and the theorem of solution existence represented by the formula ψ with interpretation I_{ψ} .

It directly follows fromGödel's theorem asserting that

$$I_{\varphi 1} \& I_{\varphi 2} \& \ldots \& I_{\varphi\mu} \& I_{\overline{\psi}} = \square \tag{0.4i}$$

implies

$$\mathrm{F} \mid\!-\, \psi \tag{0.4ii}$$

with special symbol $\mid\!-$ denoting logical (semantic) consequence. Thus, to prove that a given problem really has a solution one needs to verify equation (0.4i) above and this is therefore a pivot of further considerations.

There are some known strategies using equality calculus [11,12]. The fact that

$$I_\varphi \& I_\varepsilon = \square$$

for some formulae φ and ε immediately follows from the observation that

$$I_\varphi = \{\dots, x = \alpha\} \text{ and } I_\varepsilon = \{\dots, x \neq \alpha\}$$

with constant object α . We shall use this statement of equality calculus for direct proof of (0.4i).

The procedure of logical inference presented here is the modification of Beth's algorithm [10]. It uses the notion of semantic entailment in predicate calculus. Let φ and ψ be two well formed formulae and ψ semantically follows φ (φ |– ψ) i.e. in every interpretation where φ is true $\overline{\psi}$ is false (ψ is true).

Interpretation is called a refutation if in it φ & $\overline{\psi}$ = 1. The procedure is to look for such an interpretation in order to refute the assertion that φ |– ψ .

Let us prove (or refute) that

$$\overbrace{\forall x(P(x) \vee Q(x))}^{\varphi} \;|\!-\; \overbrace{\forall x P(x) \vee \forall x Q(x)}^{\psi}.$$

To do this we need to find an interpretation (i.e., a substitution for x) in which φ is true and ψ is false or show that there is no such interpretation.

If ψ is false then it follows that ∀ x Q(x) is false and ∀ x P(x) is false too. In its turn from the fact that ∀ x P(x) is false we can derive that ∃ a $\overline{\text{P}}$(a) and from the falsehood of ∀ x Q(x) to conclude that ∃ b $\overline{\text{Q}}$(b) with substitutions "a" and "b" for x.

Hence we get an interpretation in which ψ is false. This interpretation contains the terms

$\overline{\text{P}}$(a)
and $\overline{\text{Q}}$(b)

and produces two clauses

$$\varphi_1 = (\overline{P}(a) \vee Q(a)),$$

$\varphi_2 = (P(b) \vee \overline{Q}(b))$.

In order to make φ_1 and φ_2 both true it is sufficient to admit that Q(a) and P(b) are true. Consequently we showed that **not** (φ |– ψ) i.e. the interpretation (Table 0.3) satisfies φ and refutes ψ.

Table 0.3.

X:	"a"	"b"	t (t≠a; t≠b)
P(.)	false	true	true
Q(.)	true	false	true
$\overline{P}$()	true	false	true
$\overline{Q}$()	false	true	true

Now let

$\psi = \forall x(P(x) \vee Q(x))$

and $\varphi = \forall x(P(x)) \vee \forall x(Q(x))$.

If we assume ψ to be false we will derive $\overline{\exists x(P(x) \vee Q(x))}$. Suppose x = a. In this case there are both $\overline{P}(a)$ and $\overline{Q}(a)$. As for φ we have $\varphi = \overline{P}(a) \vee \overline{Q}(a)$. In order φ to be true, it is necessary for either $\overline{P}(a)$ or $\overline{Q}(a)$ (or both $\overline{P}(a)$ and $\overline{Q}(a)$) to be true what is in contradiction with supposition about the falsehood of ψ .Since we cannot find a refutation and have to derive φ |– ψ.

In order to find a regular procedure for seeking refuting interpretation consider the formula

F |– ψ,

where F = $\{\varphi_1,\varphi_2,\dots,\varphi_N\}$ and $\varphi_{i=\overline{1,N}}$ - are well-formed formulae of predicate calculus. In every interpretation each of the formula F becomes a propositional clause representing an expression built with atomic formulae which are connected by means of logical operators &, ∨, →, ~ and ¬ and auxiliary symbols (,), and ⇒ where & -

conjunction, $\vee$ - disjunction, $\rightarrow$ - implication, $\sim$ - equivalency (also $\leftrightarrow$ and = are used), $\neg$ - negation (often written above symbol of the formula), and $\Rightarrow$ equivalent transformation. An example of a propositional clause is

$\varphi = \overline{P}(a) \vee \overline{Q}(a)$ with constant "a".

Let $f_{i1}(x_1^{(i)},..,x_{n1}^{(i)}),...,f_{im}\,(x_1^{(i)},...,x_{nm}^{(i)})$ be the atomic formulae used for writing φ_i. Our task is to find such values for variables $x_1,...,x_n$ which provide truth of every well-formed formula φ_i and falsehood of ψ. We admit that F and ψ may contain common atomic formulae or their negations. This task may be formulated as follows: to find an interpretation I (or to prove that such interpretation is impossible) which provides the truth of conjunction φ_1 & φ_2 &...&$\overline{\psi}$.

0.3.3. Rules for making atomic interpretations

We shall use the following transformations.

1. All the variables bound by different existential quantifiers should be made different by appropriate substitutions.

An example. The system of formulae

$\forall z \forall y \exists x (P(x,y) \vee Q(z))$
and $\exists z \exists x (\overline{Q}(z) \vee \overline{P}(x,z))$

is replaced by

$\forall z \forall y \exists x (P(x,y) \vee Q(z))$
and $\exists z \exists w (\overline{Q}(z) \vee \overline{P}(w,z))$.

2. Delete all the existential quantifiers by means of special notation in which

$P(\check{x})$ stands for $\exists$ x P(x);

P(□) stands for $\overline{\exists}$ x P(x);

$P(f(\check{x})$ stands for $\exists$ x P(f(x));

□ - denotes the dummy element(set);

$\check{x}$ - denotes some concrete object from x-domain;

x - is a free variable.

We will also use the designation $\tilde{Q}_i$ for predicate arguments bearing in mind that Q is a predicate symbol and i is an argument number, e.g. $\tilde{P}\tilde{1}, \tilde{P}\tilde{2}, \tilde{P}\tilde{3}$ are used for x, y, z in P(x, y, z) correspondingly.

Consider the following two formulae:

$$\exists x \forall y P(x,y) \tag{A1}$$

$$\forall x \exists y P(x,y). \tag{A2}$$

It is necessary to understand that in (A1) one deals with some concrete value of x, i.e. it is correct to replace (A1) by

$$\overline{P}(\breve{x},\square)$$

with the help of the following equivalencies:

$$\forall z(Q(z)) \sim \overline{\exists} z(\overline{Q}(z))$$

$$\exists z(Q(z)) \sim \overline{\forall} z(\overline{Q}(z)).$$

It is not difficult to reproduce the transformations for (A1) as shown below:

$$\exists x \forall y(P(x,y) \Rightarrow \exists x \overline{\exists} y \overline{P}(x,y) \Rightarrow \exists x \overline{P}(x,\square) \Rightarrow \overline{P}(\breve{x},\square).$$

As far as the formula (A2) is concerned one should bear in mind that there is a set of y-values corresponding to every possible x-value. Therefore, in order to create a correct interpretation in this case one needs to use the mechanism of "skolemization", i.e. the representation obtained according to the scheme below:

$$\forall x \exists y(P(x,y) \Rightarrow \forall x P(x, f(\tilde{P}\tilde{1}) \Rightarrow \overline{\exists} x \overline{P}(x, f(\tilde{P}\tilde{1})) \Rightarrow \overline{P}(\square, f(\tilde{P}\tilde{1})).$$

Note that in $\overline{P}(\square, f(\tilde{P}\tilde{1}))$ $\tilde{P}\tilde{1}$ is a free variable.

We also use the following common rules:

$$\forall x \forall y \ldots \forall z\ P(x,y,\ldots,z,a,\ldots,b) \Rightarrow \overline{P}(\square,\square,\ldots,\square,a,\ldots,b)$$

($P(x,y,\ldots,z,a,\ldots,b)$ does not contain "□")

$$\forall x \forall y \ldots \forall z\, P(x,y,\ldots,z,\square) \Rightarrow P(\square,\square,\ldots,\square)$$

(P contains one or more "□"). Consider some more complicated example.

$$\forall x \exists y \forall z\, P(x,y,z,a) \Rightarrow \forall x \forall z\, P(x,g(x),z,a) \Rightarrow$$
$$\Rightarrow \overline{P}(\square,g(\tilde{P}\tilde{1}),\square,a).$$

The expression (P(_,y) & Q(_)) $\{\breve{x}\}$ is a counterpart of ∃ x ∃ y ((P(x,y) & Q(x)) where $\breve{x}$ is similarly instantiated for P and Q.

Let the formula

$$P(\breve{x}, \square, \square, \breve{y})$$

be obtained by transformations. We may read this formula as follows: "there are such x and y that it is impossible to find corresponding second and third arguments for them to ensure a true value of P".

If a formula has the form $\overline{P}(\breve{x}, \square, \square, \breve{y})$ then the final part of the previous definition should be read as "... to ensure a true value of $\overline{P}$".

Thus, to ensure true value for P it is necessary to set an interpretation $I_p = \{\tilde{P}\tilde{1} \neq \breve{x} \vee \tilde{P}\tilde{4} \neq \breve{y}\}$, where I_p is the symbol of interpretation for predicate P. The general rule determines the following distinct cases:

1) $P(\breve{x},\breve{y},\ldots\breve{z},\square,\square,\ldots,\square) \rightarrow I_p = \{\tilde{P}\tilde{1} \neq \breve{x} \vee \tilde{P}\tilde{2} \neq \breve{y} \vee \ldots \vee \tilde{P}_k \neq \breve{z}\}$

From this for $I_{\overline{p}}$ we can obtain:

$$I_{\overline{p}} = \{\tilde{P}\tilde{1} = \breve{x}, \tilde{P}\tilde{2} = \breve{y},\ldots,\tilde{P}_k = \breve{z}, \tilde{P}_{k+1} = \tilde{P}_{k+2} = \ldots = \tilde{P}_{k+N} = ALL\}$$

"ALL" represents every valid element from the corresponding domain; "," is equivalent to "&"-operator. Some correlation is needed when there is a functor in the formula's representation.

Suppose, there are two formulae

$$P(\breve{x}, f(\tilde{P}\tilde{3}), \square)$$

and $Q(g(\tilde{Q}\tilde{3}), f(\tilde{Q}\tilde{3}), \square)$

where f(.) and g(.) are Scolem functions.

Let us try to solve two equations, P = 1 and Q = 1. For the first one we obtain

$$\tilde{P}1 \neq \breve{x} \vee \tilde{P}3 = v, \tilde{P}2 \neq f(v).$$

For the second equation we obtain:

$$(\tilde{Q}\tilde{3} = v, (\tilde{Q}\tilde{2} \neq f(v) \vee \tilde{Q}\bar{1} \neq g(v))),$$

where v is a free variable.

The origin of the solution of the equation

$$Q\ (g(\tilde{Q}\tilde{3}), f(\tilde{Q}\tilde{3}), \square) = 1$$

may be explained as follows. Obviously, in supposition for Q to be true and $\tilde{Q}\tilde{3}$ = v (where v can take every possible value) it is inadmittable that $\tilde{Q}\tilde{2}$ = f(v) and $\tilde{Q}\bar{1}$ = g(v) simultaneously since in this case $\tilde{Q}\tilde{3}$ cannot be equal to v and this directly leads to a contradiction.

In the first equation, since $\tilde{P}\bar{1}$ = x̆ leads to $\tilde{P}\tilde{3}$ = □ then there is no valid value of $f(\tilde{P}\tilde{3})$ in such an interpretation. By analogy, for $\overline{P}$ and $\overline{Q}$ we obtain the following interpretations:

$$I_{\bar{p}} = \{\tilde{P}\tilde{1} = \breve{x}, \tilde{P}\tilde{3} = v, \tilde{P}\tilde{2} = f(v)\}$$

$$I_{\bar{Q}} = \{\tilde{Q}\tilde{3} = v, \tilde{Q}\tilde{1} = g(\tilde{Q}\tilde{3}), \tilde{Q}\tilde{2} = f(\tilde{Q}\tilde{3})\}$$

2) $P(\breve{x}, \breve{y}, \ldots \breve{z},) \rightarrow I_p = \{\tilde{P}\tilde{1} = \breve{x}, \tilde{P}\tilde{2} = \breve{y}, \ldots, \tilde{P}_m = \breve{z}\}$

3) $P(\square, \square, \ldots, \square) \rightarrow I_p = \square$ (an empty interpretation).

The interpretation $\tilde{P}\tilde{1} \neq \square \vee \tilde{P}\tilde{2} \neq \square \vee \ldots \vee \tilde{P}m \neq \square$ does not hold in this case. From the opposite supposition (for example, $\tilde{P}\tilde{1} \neq \square$, i.e. $\tilde{P}\tilde{1}$= x̆ for definite x) we derive a contradiction.

$I_{\overline{P}}$ in this case has the form

$$I_{\bar{p}} = \{\tilde{P}\tilde{1} = ALL, \tilde{P}\tilde{2} = ALL, \dots, \tilde{P}_m = ALL\}$$

4) From $I_p = \{\tilde{P}\tilde{1} \neq \breve{x} \vee \tilde{P}\tilde{2} \neq \breve{y}\}$

we can obtain

$$I_{\bar{p}} = \{\tilde{P}\tilde{1} = \breve{x}, \tilde{P}\tilde{2} = \breve{y}\}$$

by means of De Morgan's Rule

$$\overline{I_{p1} \vee I_{p2}} \rightarrow \bar{I}_{p1} \& \bar{I}_{p2}.$$

Let there be a set of formulae $f_1, f_2, \dots, f_n$. An interpetation I satisfying all the formulae is obtained as a conjunction of individual interpretations, i.e.

$$I = I_{f1} \ \& \ I_{f2} \ \& \ \dots \ \& \ I_{fn}. \tag{0.5}$$

We especially stress the fact that all individual interpretations from (0.5) must be concerted , i.e. represent either predicates P,Q,... or their negations only. To derive a general rule of conjunction we define an atomic interpretation I in one of the forms:

(i) I = { P = $\breve{x}$ };

(ii) I = { P ≠ $\breve{x}$ };

(iii) I = □ ;

(iv) I = ALL ;

(v) I = { P = a } (a - constant symbol).

Let I_1 and I_2 be the atomic interpretations. Then their conjunction is determined by the following cases (Table 0.4).

The case with a constant is similiar to P($\breve{x}$). The Table 0.4 requires further explanation. Namely, we shall discriminate between bound and unbound occurences of variable x denoting the former as $\breve{x}$ and the latter as x. It is clear that if

$I_1 = (P \neq \breve{x})$ and $I_2 = (P = x)$ then x=$\breve{x}$ provides
$I_1 \ \& \ I_2 = \square$.

Let us recall that our main goal is to prove that

$$I_1 \& I_2 \& \dots \& I_t = \square.$$

Table 0.4.

I_1	I_2	I_1 & I_2 = I_2 & I_1
$\{ P = \breve{x} \}$	$\{ P \neq \breve{x} \}$	$\square$
$\{ P = \breve{x} \}$	$\square$	$\square$
$\{ P = \breve{x} \}$	ALL	$\{ P = \breve{x} \}$
$\{ P \neq \breve{x} \}$	ALL	$\square$
$\{ P = \breve{x} \}$	$\{ P \neq \breve{y} \}$	$\{ P = \breve{x}, P \neq \breve{y} \}$
$\{ P = \breve{x} \}$	$\{ P = \breve{y} \}$	$\{ P = \breve{x}, P = \breve{y} \}$
ALL	ALL	ALL

So, from considerations above it directly follows that if some interpretations contain unbound variables then one has to find such substitutions for these variables which result in obtaining ' $\square$ ' for their conjunction. If, for example, I_k contains the unbound variable x then it may produce a number of individual interpretations by binding the variable x with concrete values $\breve{x}_1, \breve{x}_2, \ldots,$ etc. Fortunately, the set of these values is directly obtained as a result of building interpretations for an initial set of formulae.

Let us consider one complete example illustrating the proof on the basis of the atomic interpretations.

An example. Let us prove that the formula

$$\Psi = \exists z(P(a, \boldsymbol{z}))$$

may be derived from

$$\varphi = \forall x \exists y(P(x,y)).$$

First of all, we create an interpretation for the formula φ :

$$\text{(i)}\, \varphi = \forall x \exists\, y(P(x,y)) \Rightarrow \forall x\, P(x, f(\tilde{P}\tilde{1})) \Rightarrow \overline{\exists} x\, \overline{P}(x, f(\tilde{P}\tilde{1})) \Rightarrow$$
$$\Rightarrow \overline{P}(\square, f(\tilde{P}\tilde{1}))$$

where $\tilde{P}\tilde{1}$ is a free variable.

Then we create an interpretation for the formula $\overline{\Psi}$:

(ii) $\overline{\exists} z\, P(a,z) \Rightarrow P(a,\square)$

where "a" is a symbolic constant.

From (i) we obtain

$$\overline{P}(\square, f(\tilde{P}\tilde{1})) \rightarrow I^{1}_{\bar{p}} = \{\tilde{P}\tilde{2} \neq f(\tilde{P}\tilde{1})\}.$$

From (ii) we obtain

$$P(a,\square) \rightarrow I^{2}_{p} = \{\tilde{P}\tilde{1} \neq a\} \text{ and } I^{2}_{\bar{p}} \;=\; \{\tilde{P}\tilde{1} = a, \tilde{P}\tilde{2} = ALL\}.$$

Thus, we derive

$$I^{1}_{\bar{p}} \& I^{2}_{\bar{p}} = \square \ (i.e.,\ \tilde{P}\tilde{2} \neq f(\tilde{P}\tilde{1}) \& \tilde{P}\tilde{2} = ALL \rightarrow \square).$$

On the other hand, it follows from $I^{1}_{p} = \{\tilde{P}\tilde{1} = ALL, \tilde{P}\tilde{2} = f(\tilde{P}\tilde{1})\}$ that $I^{1}_{p} \& I^{2}_{p} = \square$ (since $\tilde{P}\tilde{1} = ALL \& \tilde{P}\tilde{1} \neq a$).

0.3.4. Rules for making complex interpretations

To build an interpretation of a complex formula means to define the variable's values providing consistency of a given formula.

Consider the next formula

$$\forall x(\exists y(S(x,y) \& M(y)) \rightarrow (\exists z(I(z) \& (Ex,z))). \qquad \textbf{(0.6a)}$$

By sequential transformations we obtain

$$\begin{aligned}
&\forall x(S(x,f(x)) \& M(f(x)) \rightarrow I(\breve{z}) \& E(x,\breve{z})) \Rightarrow \\
&\Rightarrow \forall x(\overline{S(x,f(x)) \& M(f(x))} \vee I(\breve{z}) \& E(x,\breve{z})) \Rightarrow \\
&\Rightarrow \overline{\exists} x(S(x,f(x)) \& M(f(x)) \& (\bar{I}(\breve{z}) \vee \overline{E}(x,\breve{z}))) \Rightarrow \\
&\Rightarrow S(\square_x, f(\tilde{S}\tilde{1})) \& M(f(\tilde{S}\tilde{1})) \& (\bar{I}(\breve{z}) \vee \overline{E}(\square_x,\breve{z})).
\end{aligned}$$

Our task is reduced to making an individual interpretation for the last formula

$$S(\square_x, f(\tilde{S}\tilde{1}))\&M(f(\tilde{S}\tilde{1}))\&(\bar{I}(\breve{z})\vee\overline{E}(\square_x,\breve{z})) \tag{0.6b}$$

to provide its consistency.

We see, that the formula (0.6b) has a structure of the kind

$$f_1\&f_2\&f_3,$$

where

$$f_1 \sim S(\square_x, f(\tilde{S}\tilde{1}))$$
$$f_2 \sim M(f(\tilde{S}\tilde{1}))$$
$$f_3 \sim (\bar{I}(\breve{z})\vee\overline{E}(\square_x,\breve{z})).$$

The consistency of the formula f_1 is provided by the interpretation:

$$\tilde{S}\tilde{1} = v, \tilde{S}\tilde{2} \neq f(v).$$

The consistency of the formula f_2 is provided by the interpretation:

$$\tilde{M}\tilde{1} = f(v).$$

And, finally, for f_3 we set an interpretation

$$\tilde{I}\tilde{1} = \breve{z} \vee \tilde{E}\tilde{1} = v, \tilde{E}\tilde{2} \neq \breve{z}$$

where v is a free variable.

Thus, we obtain the full interpretation for the formula (0.6b) in the form

$$I_{(2)} = \{\tilde{S}\tilde{1} = v, \tilde{S}\tilde{2} \neq f(v), \tilde{M}\tilde{1} = f(v),$$
$$(\tilde{I}\tilde{1} = \breve{z} \vee \tilde{E}\tilde{1} = v, \tilde{E}\tilde{2} \neq \breve{z})\}.$$

Let us show that from (0.6a) one may deduce the following formula [9]:

$$\overline{\exists} xI(x) \rightarrow \forall x\forall y(S(x,y) \rightarrow \overline{M}(y)) = \exists xI(x) \vee \forall x\forall y(\overline{S}(x,y) \vee \overline{M}(y)).$$

Replacing the last formula by its negation we obtain

$$\overline{\exists} xI(x)\&\overline{\forall} x\forall y(\overline{S}(x,y) \vee \overline{M}(y)) = I(\square_x)\&\exists x\exists y\overline{(\overline{S}(x,y) \vee \overline{M}(y))} =$$

$$= I(\square_x)\& S(\breve{x},\breve{y})\& M(\breve{y}).$$

For this last one we make an interpretation

$$I_{(1)} =\{\tilde{I}\tilde{1} =ALL,\tilde{S}\tilde{1} =\breve{x},\tilde{S}\tilde{2} =\breve{y},\tilde{M}\tilde{1} =\breve{y}\}.$$

It is important to note that in $I_{(1)}$ an argument $\tilde{I}\tilde{1}$ is defined for predicate $\bar{I}(\square_x)$ as it must be concerted with the interpretation I_2 which contains an individual interpretation for $\bar{I}(\breve{z})$. It is easy to convince oneself that

$$I_{(1)}\& I_{(2)} = \square.$$

In fact, when unifying two interpretations the free variable v in $I_{(2)}$ becomes **bound**, i.e. v=$\breve{x}$ and f(v)=f($\breve{x}$) which directly leads to a contradictory system:

$$\begin{cases} \tilde{S}\tilde{1} = v = \breve{x} & \\ \tilde{S}\tilde{2} \neq f(v) \rightarrow \tilde{S}\tilde{2} \neq f(\breve{x}) & \textit{from } I_{(2)} \\ \tilde{M}\tilde{1} = f(v) = f(\breve{x}) & \\ - & \\ \tilde{S}\tilde{1} = \breve{x} & \\ \tilde{S}\tilde{2} = \breve{y} & \textit{from } I_{(1)} \\ \tilde{M}\tilde{1} = \breve{y} & \end{cases}$$

where f($\breve{x}$) must simultaneously be and not be equal to $\breve{y}$. To further proceeding let us consider the formula

$$P\& M\{\square\}$$

which means that there is no such an argument x which provides truth of P(x) & M(x). Thus, if we admit that P($\breve{x}$) is true then M($\breve{x}$) should be false (i.e. $\overline{\boldsymbol{M}}$ (x) should be true) and vice versa. Therefore, if there exists an interpretation in which $\tilde{M}\tilde{1} = \breve{x}$

then to provide consistency of the formula P & M {□} we should make an interpretation

$$I = \{(\tilde{M}\tilde{1} = \breve{x} \& \tilde{P}\tilde{1} \neq \breve{x}) \vee (\tilde{M}\tilde{1} \neq \breve{x} \& \tilde{P}\tilde{1} = \breve{x})\}$$

In general, if there are the individual interpretations

$$I_1 \colon \tilde{M}\tilde{1} = \breve{x}$$
$$I_2 \colon \tilde{M}\tilde{1} = \breve{y}$$
$$\cdots\cdots$$
$$I_N \colon \tilde{M}\tilde{1} = \breve{z}$$

then to provide consistency of the formula P & M {□} we need to build an interpretation

$$\begin{aligned}&I_{N+1} \colon ((\tilde{M}\tilde{1} = \breve{x} \& \tilde{P}\tilde{1} \neq \breve{x}) \vee (\tilde{M}\tilde{1} \neq \breve{x} \& \tilde{P}\tilde{1} = \breve{x}))\&\\ &\&((\tilde{M}\tilde{1} = \breve{y} \& \tilde{P}\tilde{1} \neq \breve{y}) \vee (\tilde{M}\tilde{1} \neq \breve{y} \& \tilde{P}\tilde{1} = \breve{y}))\&\\ &\cdots\cdots\\ &\&((\tilde{M}\tilde{1} = \breve{z} \& \tilde{P}\tilde{1} \neq \breve{z}) \vee (\tilde{M}\tilde{1} \neq \breve{z} \& \tilde{P}\tilde{1} = \breve{z})).\end{aligned}$$

To simplify the interpretation for the formula $P \& \overline{M}\{□\}$ it is more convenient to write

$$I = \{(\tilde{M}\tilde{1} = x, \tilde{P}\tilde{1} = x) \vee (\tilde{M}\tilde{1} \neq x \& \tilde{P}\tilde{1} \neq x)\}$$

where x is a free variable.

By analogy, for the formula

(A0) $P \& M \& Q \& \ldots \& Z\{□\}$

we make an interpretation

$$\begin{aligned}&I = \{\tilde{P}\tilde{1} = x \& (\tilde{M}\tilde{1} \neq x \vee \tilde{Q}\tilde{1} \neq x \vee \ldots \vee \tilde{Z}\tilde{1} \neq x) \vee\\ &\tilde{M}\tilde{1} = x \& (\tilde{P}\tilde{1} \neq x \vee \tilde{Q}\tilde{1} \neq x \vee \ldots \vee \tilde{Z}\tilde{1} \neq x) \vee\\ &\cdots\cdots\\ &\tilde{Z}\tilde{1} = x \& (\tilde{M}\tilde{1} \neq x \vee \tilde{Q}\tilde{1} \neq x \vee \ldots)\}\end{aligned}$$

where x is a free variable.

Some typical generalizations of A0 may be obtained without difficulties. Let us consider, for example, the formula

$$P\& M\{\square,\square\}.$$

One may simply establish the following interpretation

$$I=\{\tilde{P}\tilde{1}=x,\tilde{P}\tilde{2}=y,(\tilde{M}\tilde{1}\neq x\vee\tilde{M}\tilde{2}\neq y)\vee\tilde{M}\tilde{1}=x,\tilde{M}\tilde{2}=y,(\tilde{P}\tilde{1}\neq x\vee\tilde{P}\tilde{2}\neq y)\}.$$

Besides, let us consider the following important example of the same kind:

$$\begin{aligned}&P\& Q\{\breve{x},\square\}\Rightarrow P(\breve{x}_{,-y})\& Q(\breve{x}_{,-y})\{\square_{y}\}\Rightarrow\\&\Rightarrow I=\{\tilde{P}\tilde{1}=\breve{x},\tilde{Q}\tilde{1}=\breve{x},(\tilde{P}\tilde{2}=y,\tilde{Q}\tilde{2}\neq y\vee\tilde{P}\tilde{2}\neq y,\tilde{Q}\tilde{2}=y)\vee\\&\vee(\tilde{P}\tilde{1}\neq\breve{x},\tilde{Q}\tilde{1}\neq\breve{x})\}\end{aligned}$$

where y is a free variable. Note that $\tilde{P}\tilde{1},\tilde{Q}\tilde{1}$ should be equally instantiated.

We shall also use the following important rules for making complex interpretations.

(A1) $P\vee Q\{I\}\rightarrow P(I)\vee Q(I_1)$

I - a common interpretation, I $\neq\square$

I_1 - is obtained from I by replacing variables

(A2) $P\vee Q\{\square\}=P(\square)\& Q(\square)$,

The following equivalences are essential in this case

(A2.1) $P(\breve{x},\square)\sim\overline{P}(\breve{x},ALL)$(since $\exists x\overline{\exists}yP(x,y)\sim\exists x\forall y\overline{P}(x,y)$)

(A2.2) $P(\breve{x},\breve{y},\square,\ \square)\sim\overline{P}(\breve{x},\breve{y},ALL,ALL)$.

From this ,

$$P(\square,\breve{x})\vee Q(\square,\breve{x})$$

gives the interpretations:

$$I_{P,Q}=\{\tilde{P}\tilde{2}\neq\breve{x}\vee\tilde{Q}\tilde{2}\neq\breve{x}\}$$

$$I_{\bar{P},\bar{Q}} = \{\tilde{P}\tilde{1} = ALL, \tilde{P}\tilde{2} = \breve{x} \vee \tilde{Q}\tilde{1} = ALL, \tilde{Q}\tilde{2} = \breve{x}\}$$

(A3) $P \& Q\{I\} \rightarrow I_P \& I_Q$

(conjunction of the individual interpretations not containing □).

(A4) Propositional formula Q without variables is replaced by $\forall x Q(x)$ (see example E0).

(A5) For disjunction D,

$$D = P(I'_P) \vee Q(I'_Q) \vee \ldots \vee R(I'_R) \quad \{I\}$$

we set an interpretation

$$I_D = I'_P \& I \vee I'_Q \& I \vee \ldots \vee I'_R \& I$$

(A6) The interpretations are built only for the formulae or their negations.

Let us consider some explanatory examples.

Example E0. Consider a propositional system

$$\varphi_1 \colon P \rightarrow S$$
$$\varphi_2 \colon S \rightarrow U$$
$$\varphi_3 \colon P$$
$$\psi \colon U.$$

To prove that $\varphi_1, \varphi_2, \varphi_3 \vdash \psi$ we introduce an equivalent system

$$\varphi'_1 \colon \forall x (P(x) \rightarrow S(x))$$
$$\varphi'_2 \colon \forall x (S(x) \rightarrow U(x))$$
$$\varphi'_3 \colon \forall x P(x)$$
$$\psi' \colon \forall x U(x)$$

(we leave the proof of correctness of such a transformation).

Now we find suitable interpretations:

$$I_{\varphi'_1} \colon P \& \bar{S}\{\square\} \rightarrow I_{\varphi'_1} = \{\tilde{P}\tilde{1} = x, \tilde{S}\tilde{1} = x \vee \tilde{P}\tilde{1} \neq x, \tilde{S}\tilde{1} \neq x\}$$

x is a free variable (unbound variable stands for "ALL" here)

$$I_{\varphi'_2}: S\&\overline{U}\{\square\} \rightarrow I_{\varphi'_2} = \{\tilde{S}\tilde{1} = x, \tilde{U}\tilde{1} = x \vee \tilde{S}\tilde{1} \neq x, \tilde{U}\tilde{1} \neq x\}$$

$$I_{\varphi'_3}: \overline{P}\{\square\} \rightarrow I_{\varphi'_3} = \{\tilde{P}\tilde{1} = ALL\}$$

($I_{\varphi'_3}$ contains an interpretation for P(x), (not for $\overline{P}(x)$) as it must be concerted according to the rule (A6) with $I_{\varphi'_1}$ which deals with P(x)).

$$I_{\overline{\psi}'}: \overline{U}(\breve{z}) \rightarrow I_{\psi'} = \{\tilde{U}\tilde{1} \neq \breve{z}\}$$

(Similarly, $I_{\psi'}$ establishes an interpretation for U(x) (not for $\overline{U}(x)$) to concert with $I_{\varphi'_2}$).

Then we obtain:

$$I_{\psi'}\&I_{\varphi'_3} = \{\tilde{U}\tilde{1} \neq \breve{z}, \tilde{P}\tilde{1} = ALL\}$$

$$(I_{\psi'}\&I_{\varphi'_3})\&I_{\varphi'_2} = \{\tilde{U}\tilde{1} \neq \breve{z}, \tilde{P}\tilde{1} = ALL, \tilde{S}\tilde{1} \neq \breve{z} \mid \textit{substitution } \breve{z} \textit{ for } x\}$$

$$I_{\psi'}\&I_{\varphi'_3}\&I_{\varphi'_2}\&I_{\varphi'_1} = \square \ (\textit{substitution} \ \ x = \breve{z})$$

Thus, ψ logically follows from $\varphi_1, \varphi_2, \varphi_3$.

Example E1. For the formulae F:

$$\exists y(S(y)\&M(y)) = S\&M\{\breve{y}\}$$

and $\forall x(\overline{P}(x) \vee \overline{M}(x)) = \overline{\exists}x\overline{(\overline{P}(x) \vee \overline{M}(x)} = P\&M\{\square\}$

to prove F |– ψ for

$$\psi = \exists z(S(z)\&\overline{P}(z)).$$

According to general strategy we must replace ψ by $\overline{\psi}$ and derive F, $\overline{\psi}$ |– (I = □).

$$\overline{\psi} = \forall z(\overline{S}(z) \vee P(z)) = \overline{\exists}z\overline{(\overline{S}(z) \vee P(z))} =$$
$$= \overline{\overline{\exists}}z(S(z)\&\overline{P}(z)) = S\&\overline{P}\{\square\}$$

Now we have :

for $S\& M\{\breve{y}\} \rightarrow I1 = \{\tilde{S}\tilde{1} = \breve{y}\& \tilde{M}\tilde{1} = \breve{y}\}$;
for $P\& M\{\square\} \rightarrow I2 = \{(\tilde{M}\tilde{1} = \breve{y}\& \tilde{P}\tilde{1} \neq \breve{y}) \vee (\tilde{M}\tilde{1} \neq \breve{y}\& \tilde{P}\tilde{1} = \breve{y})\}$;
for $S\& \overline{P}\{\square\} \rightarrow I3 = \{(\tilde{S}\tilde{1} = \breve{y}\& \tilde{P}\tilde{1} = \breve{y}) \vee (\tilde{S}\tilde{1} \neq \breve{y}\& \tilde{P}\tilde{1} \neq \breve{y})\}$.

Note, that I3 is obtained from the relation between I_p and $I_{\overline{p}}$.

Thus, if in $I_p\tilde{P}\tilde{1} \neq \breve{y}$ then in $I_{\overline{p}}\tilde{P}\tilde{1} = \breve{y}$ and vice versa.

It is easy to see that

$$I1\& I2\& I3 = \square$$

(namely what we want to conclude).

Example E2. To prove that

$F1, F2 \mid\!- G$,where
$F1{:}\, \forall x(C(x) \rightarrow (W(x)\& R(x)))$.
$F2{:}\, \exists x(C(x)\& O(x))$.
$G{:}\, \exists x(O(x)\& R(x))$.

After all necessary transformations we get

$F1{:}C\& (\overline{W} \vee \overline{R})\{\square\}$.
$F2{:}C\& O\{\breve{x}\}$.
$\overline{G}{:}O\& R\{\square\}$.

We have the following interpretations :

$I_{F2}{:}\{\tilde{C}\tilde{1} = \breve{x}, \tilde{O}\tilde{1} = \breve{x}\}$
$I_{\overline{G}}{:}\{(\tilde{O}\tilde{1} = z, R1 \neq z) \vee (\tilde{O}\tilde{1} \neq z\& \tilde{R}\tilde{1} = z)\}$

z is a free variable.

$I_{F1}{:}\{(\tilde{C}\tilde{1} = u\& \tilde{W}\tilde{1} = u\& \tilde{R}\tilde{1} = u) \vee$
$(\tilde{C}\tilde{1} \neq u\& \tilde{W}\tilde{1} \neq u\& \tilde{R}\tilde{1} \neq u)\}$

u is a free variable.

Now we obtain

$$I_{F2}\& I_{\overline{G}}|_{Z=\breve{X}}:$$
$$\{\tilde{C}\tilde{1} = \breve{x}, \tilde{O}\tilde{1} = \breve{x}, \tilde{R}\tilde{1} \neq \breve{x}\}$$

and proceed from

$$(I_{F2}\& I_{\overline{G}})\& I_{F1}|_{U=\breve{X}}:\square$$

In this example we used the substitution $\breve{x}$ for z and u.

Example E3.

From the formulae

$$\varphi_1: \forall x(E(x)\& \overline{V}(x) \rightarrow \exists y(S(x,y)\& C(y)))$$
$$\varphi_2: \exists x(P(x)\& E(x)\& \forall y(S(x,y) \rightarrow P(y)))$$
$$\varphi_3: \forall x(P(x) \rightarrow \overline{V}(x))$$

to deduce

$$\psi: \exists x(P(x)\& C(x)).$$

Perform all necessary transformations:

$$(1)\forall x(E(x)\& \overline{V}(x) \rightarrow \exists y(S(x,y)\& C(y))) \sim$$
$$\sim \forall x(\overline{E(x)\& \overline{V}(x)} \vee S(x,\breve{y})\& C(\breve{y})) \sim$$
$$\sim \overline{\exists x}(E(x)\& \overline{V}(x)\& (\overline{S}(x,\breve{y}) \vee \overline{C}(\breve{y}))$$

with an appropriate interpretation I1:

$$I1 = \{\tilde{E}\tilde{1} = x\& \tilde{V}\tilde{1} = x \vee \tilde{E}\tilde{1} \neq x\& \tilde{V}\tilde{1} \neq x\}$$

x is unbound; the disjunction $\overline{S}(x,\breve{y}) \vee \overline{C}(\breve{y})$ may not be taken into account due to the fact that $\overline{C}(\breve{y})$ is unconditionally true.

$$(2)\exists x(P(x)\& E(x)\& \forall y(\overline{S}(x,y) \vee P(y))) \sim$$
$$\sim \exists x(P(x)\& E(x)\& \overline{\exists} y(S(x,y)\& \overline{P}(y))) \sim$$

$$\sim \exists x(P(x)\& E(x)\&(S(x,_{-y})\&\overline{P}(_{-y})\{\Box_y\}))$$

$$I2{:}=\{\tilde{P}\tilde{1}=\breve{z}\&\tilde{E}\tilde{1}=\breve{z}\&(\tilde{S}\tilde{1}=\breve{z}\&\tilde{S}\tilde{2}=y\&\tilde{P}\tilde{1}=y\vee$$
$$\vee\tilde{S}\tilde{1}=\breve{z}\&\tilde{S}\tilde{2}\neq y\&\tilde{P}\tilde{1}\neq y)\}$$

with an unbound variable y and substitution $\breve{z}$ for x

$$(3)\forall x(P(x)\rightarrow\overline{V}(x))\sim\forall x(\overline{P}(x)\vee\overline{V}(x))\,{}^{\sim}\,\overline{\exists}x(P(x)\&V(x))$$

$$I3{:}=\{\tilde{P}\tilde{1}=u\&\tilde{V}\tilde{1}\neq u\vee\tilde{P}\tilde{1}\neq u\&\tilde{V}\tilde{1}=u\}$$

u is a free variable.

$$(4)\overline{\psi}\sim\overline{\exists}x(P(x)\&C(x))$$

$$I4{:}=\{\tilde{P}\tilde{1}=v\&\tilde{C}\tilde{1}\neq v\vee\tilde{P}\tilde{1}\neq v\&\tilde{C}\tilde{1}=v\}$$

v is a free variable.

To find I3 & I4 it is necessary previously to unify the unbound variables standing for the same predicate's arguments, i.e. to substitute v for u. Thus we get

$$I3\&I4=\{\tilde{P}\tilde{1}=v,\tilde{C}\tilde{1}\neq v,\tilde{V}\tilde{1}\neq v\vee$$
$$\vee\tilde{P}\tilde{1}\neq v,\tilde{C}\tilde{1}=v,\tilde{V}\tilde{1}=v\}$$

with unbound variable v.

Further we find

$$(I3\&I4)\&I2.$$

To do it we bind variables v and y with $\breve{z}$:

$$I3\&I4\&I2{:}=\{\tilde{P}\tilde{1}=\breve{z}\&\tilde{E}\tilde{1}=\breve{z}\&\tilde{S}\tilde{1}=\breve{z}\&$$
$$\&\tilde{S}\tilde{2}=\breve{z}\&\tilde{C}\tilde{1}\neq\breve{z}\&\tilde{V}\tilde{1}\neq\breve{z}\}.$$

And finally we obtain I3 & I4 & I2 & I1:= □ using substitution $\breve{z}$ for unbound variable x in I1.

0.3.5. Remarks on correctness of the procedure

The grounds of the considered approach to proving deducibility of some formula ψ from a given set F of formulae are connected with the well-known Gedel completeness theorem.

Therefore, we should prove correctness of the rules (A0 - A6) making complex interpretations. To be more precise we resrict our considerations by the rules (A0), (A1), (A2) and (A5) since the other rules may be proved by analogy.

Atomic interpretations are considered to be true by definition.

Let us consider the rule (A0):

$$P\&Q\{\Box\} \rightarrow I = \{\tilde{P}\tilde{\bar{1}} = x, \tilde{Q}\tilde{\bar{1}} \neq x \vee \tilde{P}\tilde{\bar{1}} \neq x, \tilde{Q}\tilde{\bar{1}} = x\}.$$

Suppose the opposite is true, i.e. there is a term t such that

P(t) = 1, Q(t) = 1.

Since

$$P\&Q\{\Box\} \sim \overline{\exists}x(P(x)\&Q(x)) \sim \forall x(\overline{P}(x) \vee \overline{Q}(x))$$

then it may be derived that $\overline{P}(t) \vee \overline{Q}(t)$ is true which leads to a contradiction, that is

$$(\overline{P}(t) \vee \overline{Q}(t))\&P(t)\&Q(t) = \Box.$$

On the other hand, assume that $\tilde{P}\tilde{\bar{1}} \neq x, \tilde{Q}\tilde{\bar{1}} \neq x$. From this supposition one may derive

$$P(t_1) = 1, Q(t_2) = 1, t_1 \neq t_2.$$

It gives

$$(\overline{P}(t_1) \vee \overline{Q}(t_1))(\overline{P}(t_2) \vee \overline{Q}(t_2)) \cdot P(t_1) \cdot Q(t_2) =$$
$$= \overline{Q}(t_1) \cdot P(t_1) \cdot Q(t_2) \cdot \overline{P}(t_2)$$

and, therefore,

$\tilde{P}\tilde{1} = t_1, \tilde{Q}\tilde{1} \neq t_1$ and $\tilde{P}\tilde{1} \neq t_2, \tilde{Q}\tilde{1} = t_2$.

It is easy to see that the result obtained is fully concerted with the rule (A0). The generalization of this proof may be done analogically.

Now consider rule (A1).

Suppose, $P \vee Q\{\breve{x}\}$. This is equivalent to $\exists x(P(x) \vee Q(x))$ from which one may derive $\exists xP(x) \vee \exists xQ(x)$. Therefore,

$$\exists xP(x) \vee \exists yQ(y) \rightarrow P(\breve{x}) \vee Q(\breve{y}).$$

Now consider an interpretation $I = \{\breve{x}, \breve{y}, \ldots, \breve{z}, \square \cdots, \square\}$ Rule (A1) gives

$$\begin{aligned}
&P \vee Q\{\breve{x}, \square\} \sim \\
&\exists x \overline{\exists} y(P(x,y) \vee Q(x,y)) \sim \\
&\exists x \forall y(\overline{P}(x,y) \& \overline{Q}(x,y)) \sim \\
&\exists x(\forall y \overline{P}(x,y)) \& \forall z \overline{Q}(x,z)) \rightarrow \\
&\rightarrow \exists x(\overline{\exists} yP(x,y) \vee \overline{\exists} zQ(x,z)) \sim \\
&\sim \exists x \overline{\exists} yP(x,y) \vee \exists x \overline{\exists} zQ(x,z) \sim \\
&\sim \exists x \overline{\exists} yP(x,y) \vee \exists w \overline{\exists} zQ(w,z)) \rightarrow \\
&\sim\rightarrow P(\check{x}, \square) \vee Q(\check{w}, \square).
\end{aligned}$$

For the rule (A2) we obtain

$$\begin{aligned}
&P \vee Q\{\square\} \sim \overline{\exists} x(P(x) \vee Q(x)) \sim \\
&\sim \forall x\overline{(P(x) \vee Q(x))} \sim \forall x\overline{P}(x) \& \overline{Q}(x) \rightarrow P(\square) \& Q(\square).
\end{aligned}$$

The rules (A3) and (A4) may be proved without difficulties. Now consider the rule (A5). Suppose,

$$D = P(\check{x}_1, \check{y}_1, \ldots, \check{z}_1) \vee Q(\check{x}_2, \check{y}_2, \ldots, \check{z}_2)\{\check{u}\}.$$

It is equivalent to

$$\begin{aligned}
&\exists x_1 \exists y_1 \ldots \exists z_1 \exists x_2 \exists y_2 \ldots \exists z_2 \exists u((P(x_1, y_1, \ldots, z_1, u) \vee \\
&\vee Q(x_2, y_2, \ldots, z_2, u)) \sim
\end{aligned}$$

$$\begin{aligned}&\sim\exists x_1\exists y_1\dots\exists z_1\exists u(P(x_1,y_1,\dots,z_1,u)\vee\\ &\vee\exists x_2\exists y_2\dots\exists z_2\exists u(Q(x_2,y_2,\dots,z_2,u))\\ &\rightarrow I_p\&I\vee I_q\&I.\end{aligned}$$

By induction (A5) may be proved for n > 2 formulae.

0.4. Finding a solution by means of theorem proving

0.4.1. Using logic programming

It is rather natural to apply theorem proving techniques to problem solving. As was said before there are different problems one deals with when solving problems: (i) proving solution existence and (ii) finding a solution. The theorem proving technique is noteworthy for its combining the above mentioned problems, i.e to prove the theorem of solution existence means (in some way) to find a solution. The most effective (from a practical viewpoint) realization of this technique is associated with the programming language Prolog [13,14]. In view of the fact that there are a lot of books concerning Prolog and related issues we shall outline the whole approach in a quite general manner.

Prolog uses Horn's notation for well formed formulae with general representation (0.7):

$$A_1(x_1,\dots,x_{n1})\& A_2(y_1,\dots,y_{n2})\&\dots\& A_m(z_1,\dots,z_{nm})\rightarrow B(u_1,\dots,u_{nk})\quad \textbf{(0.7a)}$$

where$A_1(\dots),\dots,B(\dots)$ are the atomic formulae.

Representation **(0.7a)** is called a clause with the head B(.) and body $A_1(\dots),\dots,A_m(\dots)$. A clause may be either a fact, a rule, or a goal.

A fact is a clause without body. A goal is a clause without head. A rule is a clause in general form (0.7a).

The programming representation of a Horn-clause (0.7a) corresponds to the following clause-form:

$$
\begin{aligned}
&B(u_1,\ldots,u_{nk}):-A_1(x_1,\ldots,x_{n1}),\\
&A_2(y_1,\ldots,y_{n2}),\\
&\ldots\\
&A_m(z_1,\ldots,z_{nm}).
\end{aligned}
\tag{0.7b}
$$

with symbol ":-" standing for "→" ,meaning "implies".

An execution of a rule consists in proving its head formula. This process results in obtaining a suitable interpretation for the formulae's arguments. To prove a head formula of a clause Prolog subsequently proves every formula in the body of the clause. Thus, to prove $B(u_1,...,u_{nk})$ it starts with proving $A_1(x_1,...,x_{n1})$. After proving $A_1(x_1,...,x_{n1})$ Prolog tries to prove $A_2(...)$, etc. There must be corresponding clauses (rules and facts) for every atomic formula, besides so-called standard formulae (for example, those connected with input-output and file processing). When proving an atomic formula Prolog selects the first alternative rule for this formula and executes it. If execution fails Prolog tries to execute the second rule, etc. If there are no more alternatives Prolog returns to the previous atomic formula and tries to prove it using other alternative rules. The whole process resembles looking for a path in a maze. The atomic formulae correspond to the grounds in the labirinth and rules (facts) correspond to the different ways out of the grounds. Consider the following example. Let there be given the Prolog-program fragment to calculate the area of a triangle shown in Fig.(0.5)

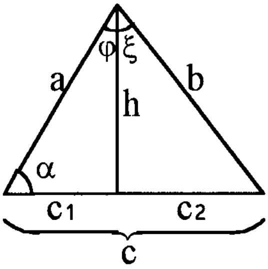


Fig 0. 5.

```
/*(R1)*/ S(_,_,C1,C2,H,_,_,_,_,Square):-
bound (H),
bound (C1),
bound (C2),
Square = 1/2 * H * (C1 + C2).

/*(R2)*/ S(A,_,C1,C2,_,ALP,_,_,_,Square):-
bound (A),
bound (C1),
bound (C2),
bound (ALP),
Square = A * (C1 + C2) * SIN(ALP) * 1/2.

/*(R3)*/ S(A,B,_,_,_,_,FI,XI,Square):-
bound (A),
bound (B),
bound (FI),
bound (XI),
Square = A * B * SIN(FI + XI) * 1/2.
... (etc.)
```

Here bound (X) is standard predicate to test if a variable X is bound or free (i.e. hasn't a value). Here in this example are shown three rules R1, R2 and R3 for the atomic formula S(.) with arguments A, B, C1, C2, H, ALP, FI, XI corresponding to a, b, c1, c2 , h, α , φ , ξ in Fig. 0.5. Argument Square stands for triangle's area.

Consider, for example, rule (R1). When proving predicate S(.) using this rule Prolog first tests that variables H, C1 and C2 are bound by some values. In the case of this test being successful an area is simply found as a product 1/2*H*(C1 + C2). If rules R1 fails (because one of the variables H, C1 and C2 is unbound) Prolog will try the next rule (R2) in which it is expected that four variables A, C1, C2 and ALP are bound. If R2 also fails Prolog will try R3 and so on.

The theorem of solution existence is written as the goal-clause of a Prolog program, for example,

```
Goal
```

```
A = 5,
B = 10,
FI = 20,
XI = 10,
S(A, B, C1, C2, H, ALP, FI, XI, Square),
write("Square =",Square).
```

The theorem of solution existence corresponding to this goal is the following

$$\Psi = ((\forall C1)(\forall C2)(\forall H)(\forall ALP)(\exists Square))(S(5,10,C1,C2,H,ALP,$$
$$20,10,Square)).$$

From the previous section one may conclude that to prove deducibility

$$F \vdash \Psi$$

one should show that

$$F,\overline{\Psi} \vdash \square$$

i.e. that a set of formulae $\{F,\overline{\Psi}\}$is contradictory. Making a negation of Ψ we obtain:

$$\exists C1 \exists C2 \exists H \exists ALP \forall Square\ \overline{S}(5,10,C1,C2,H,ALP,20,10,Square)$$

or after normalizing and skolemization

$$\overline{S}(5,10,c1,c2,h,alp,20,10,Square).$$

So, if we find an interpretation I providing the truth of F and falsehood of $\overline{\Psi}$ it would contain a required value of variable "Square" we are interested in.

To prove a goal-clause Prolog uses a kind of linear resolution strategy. One may see that every Horn-clause

$$A(\ldots):-B(\ldots),C(\ldots),\ldots,D(\ldots)$$

is equivalent to disjunction

$$\overline{B}(\ldots)\vee\overline{C}(\ldots)\vee\ldots\vee\overline{D}(\ldots)\vee A(\ldots).$$

When proving atomic formula $A(...)$ one in fact attempts to refute its negation, i.e. $\overline{A}(...)$. It is as in our case with the goal S. Suppose, there are two formulae:

$\overline{A}(...)$ (*)

and $\overline{B}(...) \vee \overline{C}(...) \vee ... \vee \overline{D}(...) \vee A(...)$ (**)

and, besides, the arguments in $\overline{A}(...)$ and in A are indiscernible (it may be done by means of the mechanism of unification briefly outlined below). Then, logical derivation from (*) and (**) is a formula

$\overline{B}(...) \vee \overline{C}(...) \vee ... \vee \overline{D}(...)$ (***)

which is called a resolvent. Since one needs to obtain an empty formula (□) then at the next step one takes the resolvent above and the rule for B(.), e.g.

B(.):- E(.),
F(.),
...
G(.).

and tries to obtain their resolvent and so on. It is easy to notice that only facts really contract current resolvent. It therefore means that proving actually consists of reducing a given formula to some known facts. This is a deductive way of reasoning. The other reason for a resolvent being equal to "□" is the falsehood of every atomic formula in representation (***).

A peculiarity of Prolog consists in availability of a set of rules for every atomic formula. To select a suitable rule Prolog first of all performs the unification of a given atomic formula belonging to current resolvent and the head formula of the corresponding rule. Unification means valid substitution for a formula's arguments to provide their equality in both formulae participating in resolution. The rules of unification are summarized by Table 0.5 (for more details see [9,13]).

Returning to our example, when unifying

Table 0.5.

First argument	Second argument	Representation after unification for both arguments
X (free variable)	Y (free variable)	$(X=Y)^{Y}$
X (free variable)	a (constant)	$(X=a)^{a}$
X (free variable)	f(a) (functor; a-constant)	f(a) (X=f(a))
X (free variable)	f(z) (functor; z-free variable)	f(z) (X=f(z))
a (constant)	b (constant)	fail
a (constant)	f(b) (functor; b-constant)	fail
f(x) (functor; x-free variable)	f(y) (functor; y-free variable)	$(Y=X)^{f(y)}$

$S(\_,\_,C1,C2,H,\_,\_,\_,\_,Square^{(1)})$ and
$\overline{S}(5,10,c1,c2,h,alp,20,10,Square^{(2)})$

there would be obtained the next matching:

$$C1 = c1$$
$$C2 = c2$$
$$H = h$$
$$Square^{(1)} = Square^{(2)}$$

An underlying symbol (_) stands for "indifferent" variable matching for any valid argument of a formula. The resolvent P_1 is the following

$$P_1 = (\overline{bound(H)} \vee \overline{bound(C1)} \vee \overline{bound(C2)} \vee$$
$$\vee \overline{Square^{(2)} = 1/2 * H * (C1 + C2)}).$$

It is impossible to cut, for example, atomic formula $\overline{bound(H)}$ from P_1 because predicate $bound(H)$ is not given . Therefore, Prolog will try rule (R2) (with the same result) and then - R3. For R3 Prolog obtains the following resolvent:

$$P_3 = \overline{bound(A)} \vee \overline{bound(B)} \vee \overline{bound(F1)} \vee \overline{bound(X1)} \vee$$
$$\vee \overline{Square^{(2)} = A * B * SIN(F1 + X1) * 1/2}.$$

Now, on the contrary, the following takes place:

$$\overline{bound(A)} = "false",$$
$$\overline{bound(B)} = "false",$$
$$\overline{bound(F1)} = "false",$$
$$\overline{bound(X1)} = "false".$$

As far as the formula

$$Square^{(2)} = A * B * Sin(F1 + X1) * 1/2$$

is concerned it is evident that for

A = 5; B = 10; FI = 20; XI =10,

$Square^{(2)}$ would be equal to 25/2. That last value of $Square^{(2)}$ is the only suitable one to provide

$$\overline{Square^{(2)} = A * B * Sin\ (F1 + X1) * 1/2} = "false".$$

Therefore, P_3 =("false")   what should be obtained) and the interpretation found is a refutation for the set of formulae

$$\{R1, R2, R3, \overline{S}, bound(A), bound(B), bound(F1), bound(X1)\}.$$

Hence, Prolog really proved the goal-clause and found a suitable interpretation representing a solution of the initial problem.

0.4.2. Combining proving and modeling techniques

One of the drawbacks of the theorem proving approach to solving problems is the static representation of a problem. Even excluding non-traditional temporal logic, the possibilities of modern universal simulation systems (such as GPSS, Simula, GASP, SLAM et. al.) are of great importance for problem solving. In this section we consider combined simulation and proving techniques. There are some known versions of Prolog utilizing this peculiarity, for example T-Prolog, developed in Hungary.

The basis for further considerations is the notion of generalized Horn clause with the following particular example:

P:- Q, R, & T, & W, Z.

where in addition to atomic formulae Q, R, Z are given two modeling programs (processes) T and W with prefix "&". To prove P in this example means subsequently to prove Q, R to fulfil T and W and then to prove Z. Fulfilment of processes T and W is understood in the sense of their termination in a given final state (states). Otherwise, Prolog interprets termination of a simulation process as a failure. Since processes T and W are developing in time then P is undefined until their termination. It means in its turn that all processes dependent on P cannot be initiated. Thus, in the case of endless processes (either T or W) it is impossible to say if P is true or false as follows from the well-known "Halting Problem" of A. Turing [7, 10].

There are two possibilities of realization of processes T and W. The first one is the realization of a simulating process and the second one is the realization of a planning process. The last case, evidently, is the most difficult. To proceed with it let us introduce:

- P^{in} and P^{fi} - initial and final states of the simulated system;
- U_i (i = 1,I) - a set of operators applied to the system states;
- predicate formulae Z_i corresponding to operators U_i; the true value of Z_i (P_j) is the necessary pre-condition for U_i to be valid operator for the state P_j.

The planning process is defined as a finite sequence K of operators providing transformation

$$P^{in} \xrightarrow{K} P^{fi}.$$

Let us consider some formalized subproblems concerning building planning sequence.

0.4.2.1. Making a planning sequence when pre-conditions are omitted

Suppose that

$$P^{in} = \left\langle P_1^{in}, P_2^{in}, \ldots, P_m^{in} \right\rangle, P^{fi} = \left\langle P_1^{fi}, P_2^{fi}, \ldots, P_m^{fi} \right\rangle,$$
$$P_l^{in}, P_l^{fi} \in \{0,1,*\}, l = \overline{1,m}$$

i.e. we deal with ternary-vectors representing states of the simulated system. We should interpret $P_j = *$ as an indifferent (from the viewpoint of planning goals) element of corresponding state vector.

Every operator U_s can be represented as a vector

$$U_s = < U_1^s, U_2^s, \ldots, U_m^s >$$

with elements $U_j^s \in \{0,1,*\}$.

If $U_j^S = *$ then it follows that U^S has no influence on the element P_j of a state vector. Thus, the task is formulated as follows: there are given P^{in}, P^{fi}, $\{U_S \mid s=\overline{1,S}\}$. It is necessary to find a sequence K of operators providing mapping $P^{in} \xrightarrow{K} P^{fi}$.

Let us continue with the algorithm solving this task. Afterwards we shall give a necessary proof of the algorithm's correctness.

Algorithm for making planning sequence K

Step 1. Introduce auxiliary vector μ:

$$\mu = < \mu_1, \mu_2, \dots, \mu_n >$$
$$\mu_i = P_i^{fi}, i = \overline{1,m}$$
$$P^{in} = < P_1^{in}, P_2^{in}, \dots, P_m^{in} >, P^{fi} = < P_1^{fi}, P_2^{fi}, \dots, P_m^{fi} >$$
$$P_1^{in}, P_1^{fi} \in \{0, 1, *\}, l = \overline{1,m}$$

Set Z = 1, K = ∅.

Step 2. If

$$(\forall l)(\mu_1 \in \{*, P_l^{in}\})$$

then stop; otherwise determine the subset $\tilde{U}^z$ of operators U_β ($U_\beta \notin$ K) satisfying the next selection rules:

$$\text{(B1)}(\forall l)((\mu_1 \neq *) \to (U_l^\beta \in \{*, \mu_l\}))$$
$$\text{(B2)}\ (\exists l)((\mu_1 \neq *) \& (U_l^\beta = \mu_l)), l = \overline{1,m}$$

where → is an implication and & denotes conjunction. (The conditions (B1), (B2) are explained in the proof of the algorithm's correctness).

Step 3. If $\tilde{U}^z = \varnothing$ then stop with general failure. If $\tilde{U}^z \neq \varnothing$ then having in mind reduction of the set K length, select from $\tilde{U}^z$ (if $|\tilde{U}^z| > 1$) operator U_γ providing maximum value of

$$\chi(U_\gamma) = \chi^1(U_\gamma) + \chi^2(U_\gamma),$$

where $\chi^1(U_\gamma)$ is the number of elements $U_l^\gamma \in U_\gamma$ satisfying (B1); $\chi^2(U_\gamma)$ is the number of those elements $U_l^\gamma \in U_\gamma$ satisfying the more rigid condition:

$$((U_l^\gamma = \mu_l \neq *) \& (\mu_l \neq P_l^{in}).$$

Obviously, the selected operator U_γ provides the fastest approach to the final state pfi .

Step 4. The operator U_γ selected at the previous step is assigned to the position Z from the end of K. Set Z= Z+1 and $\mu = \mu \circ U$, where operation '∘' is given by the Table 0.6. Then go to step 2.

The correctness of the algorithm is explained by the following

a) The last operator in K(K ≠ ∅) must not "distort" the final state P^{fi}. It means that this operator provides a valid setting of elements in P^{fi}, i.e. those elements which must be set to unity cannot be set to zero and those elements which must be set to zero cannot be set to unity. This directly follows from the rule (B1).

b) Since we exclude self-transformations (i.e. those in the form P'⇒ P", P' = P"), then the selected operator must ensure these requirements are met (the rule (B2)).

c) Finaly, the points a),b) remain justified for every intermediate state from the end of the transitions chain

$$P^{in} \Rightarrow P^1 \Rightarrow P^2 \Rightarrow \cdots \Rightarrow P^{fi}$$

when this state is considered as final after excluding those elements of the state vector which are set by the following operators.

Table 0.6.

meaning of μ_l	**meaning of U_l**	**meaning of a result $\mu_l = \mu_l \circ U_l$**
0	0	*
0 1	1 0	Invalid combinations which are cut by the rules (B1), (B2)
1	1	*
0	*	0
1	*	1
*	1	*
*	0	*
*	*	*

Example.

Suppose,

$P^{in} = < 1\ 1\ 0\ 0\ 1\ 1 >$

$P^{fi} = < 1\ 0\ 0\ 1\ 1\ 0 >$

$U_1 = < 1\ 1\ 0\ *\ *\ * >$

$U_2 = < *\ *\ *\ 1\ 0\ 1 >$

$U_3 = < *\ *\ 0\ *\ *\ 0 >$

$U_4 = < *\ *\ 1\ *\ 1\ * >$

$U_5 = < *\ 0\ 1\ *\ *\ 1 >$.

At the first step only one operator U_3 satisfies the rules (B1, B2). Therefore, we set K = <U_3 >. As a result we find the state P' preceding the final state P^{fi}, i.e.

$P' = <1\ 0\ \ *\ 1\ 1\ * >$.

P' is obtained by replacing those elements in P^{fi} which are set (in "0" or in "1") by the operator U_3 . Consider now the state P' as a final one. This time operator U_5 is a candidate to K, i.e. K = < U_3 , U_5 > and P''= <1 * * 1 1 *>. For P'' we may select operator U_4 :

$K = < U_4 , U_5, U_3 >$

and obtain P''' = < 1 * * 1 * * >.

Finally, we include in K operator U_2 and obtain:

$P^{in} \in P^{iv} = < 1\ *\ *\ *\ *\ * >$

$K_{res} = < U_2 ,U_4 , U_5, U_3 >$.

0.4.2.2. Making planning sequence with pre-conditions given by intervals

Suppose that every operator U_i is connected to the condition Z_i given by an interval, e.g. Z_i =<1 * 0 * 1 1 *>. From considerations above, it follows that Z_i determines some subset of problem states, i.e.

$< 1\,0\,0\,0\,1\,1\,0 > < 1\,1\,0\,0\,1\,1\,0 >$

$< 1\,0\,0\,1\,1\,1\,0 > < 1\,1\,0\,1\,1\,1\,0 >$

$< 1\,0\,0\,0\,1\,1\,1 > < 1\,1\,0\,0\,1\,1\,1 >$

$< 1\,0\,0\,1\,1\,1\,1 > < 1\,1\,0\,1\,1\,1\,1 >$.

Now every operator may be represented as a pair $<U_i, Z_i>$ with condition Z_i necessary for initiating the operator. Denote by ρ_i the subset of resulting states achieved by the execution of U_i, i.e.,

$$\rho_i = U_i(Z_i).$$

For example, if $Z_i = < 1 * 0 * 1\ 1 * >$ and $U_i = < 0\ 1\ 1\ 0 * * 0 >$ then $\rho_i =$ $= < 0\ 1\ 1\ 0\ 1\ 1\ 0 >$. It means that one always gets only the state $< 0\ 1\ 1\ 0\ 1\ 1\ 0 >$ as a result of the application of operator U to every state defined by Z_i.

It is obvious that the intersection

$$\pi_{ij} = \rho_i \cap Z_j$$

defines a subset of states obtained by the execution of U_i which suit Z_j , i.e. provide initiation of U_j.

Let there be given a system of operators as shown below

$< Z_1 , U_1 > = (< 0\ 1 * >, < 1 * 1 >)\ \rho_1 = < 1\ 1\ 1 >$

$< Z_2 , U_2 > = (< * 0 * >, < * 0\ 0 >)\ \rho_2 = < * 0\ 0 >$

$< Z_3 , U_3 > = (< * 1\ 0 >, < 1\ 1 * >)\ \rho_3 = < 1\ 1\ 0 >$

$< Z_4 , U_4 > = (< * * 1 >, < * 0\ 1 >)\ \rho_4 = < * 0\ 1 >$

$< Z_5 , U_5 > = (< 0\ 0 * >, < 1\ 1 * >)\ \rho_5 = < 1\ 1 * >$

$< Z_6 , U_6 > = (< 1\ 1 * >, < 0 * 1 >)\ \rho_6 = < 0\ 1\ 1 >$.

Suppose, it is required to find the planning sequence K providing mapping

$$P^{in} = < 110 > \overset{K}{\Rightarrow} P^{fi} = < 111 >.$$

Considering the initial state $P^{in} = < 1\ 1\ 0 >$ one may select either operator U_3 or U_6 . Thus, if K is represented as

$$K = K^{start} \;||\; K^{inter} \;||\; K^{finish}$$

where K ^{start} - starting subsequence of K,

and K $^{inter(mediate)}$ - intermediate subsequence of K,

and K ^{finish} - ending subsequence of K,

and | | - concatenation

then it may be written that

$$K^{start} = (U_3(P^{in}) \vee U_6(P^{in})) = (<110>_{U_6} \vee <011>_{U_6}).$$

When using this special notation for K ^{start} one should bear in mind that instead of operator symbols we use corresponding state vector representations. Thus, for K ^{finish} we have:

$$K^{finish} = (U_1 \vee U_5) = (<01*> U_1 \vee <001> U_5).$$

The last disjunction has been obtained in the supposition that the final state was achieved after applying either operator U_1 or U_5 . In the former case one may draw a conclusion that the second to last state was <0 1 *> since U_1 (< 0 1 *>)=<1 1 1>. In the latter case the analogous considerations are correct for operator U_5 . Our idea lays in the following.

(R1) To build both sequences K ^{start} and K ^{finish} by turns until it is possible (all necessary premises for this are discussed below).

(R2) If there is a common state in K^{start} and K^{finish} then this state connects both subsequences. Thus, in the example above we find such a state: $< 0\ 1\ 1 >_{U_6}$ in K^{start} and $< 0\ 1\ * >_{U_1}$ in K^{finish}. Therefore in our example we obtain K = < U_6 , U_1 >.

Consider again K^{start} = (<1 1 0> $_{U_3}$ ∨ <0 1 1>$_{U_6}$). Since the state < 1 1 0 > $_{U_3}$ is equal to P^{in} then this state may be deleted. It gives us the first concretization of (R1), namely:

(R1.1). If K^{start} has a duplicate of the state obtained earlier then the younger copy should be deleted.

In our example it gives

$$K^{start} = (<011>_{U_6}).$$

(R1.2). If the disjunction of the states in K^{start} becomes empty then there is no valid solution for an initial task.

Proof. Supposition that in terms of assertion (R1.2) there is a common state, connecting K^{start} and K^{finish} directly leads to a contradiction.

From $K^{start} = (< 0\ 1\ 1 >_{U_6})$ one may obtain a new K^{start} with representation

$$K^{start} = (U_1(<011>_{U_6}) \vee U_4(<011>_{U_6})) =$$
$$= (<111>_{U_6U_1} \vee <001>_{U_6U_4}).$$

The same considerations enable us to obtain new representation of K^{finish}, i.e.

$$K^{finish} = (<11*>_{U_1U_6} \vee <0*1>_{U_4U_5}).$$

It gives us a new solution to the problem, since there is another common state in K^{finish} $(< 0 * 1 >_{U_4U_5})$ and in K^{start} $(< 0\ 0\ 1 >_{U_6\ U_4})$; i.e.

$$K = < U_6U_4U_5 >.$$

(R1.3). If for some state (operator) in K^{start} (K^{finish}) there is no valid following state (i.e. $\pi_{ij} = \varnothing$) then this state (operator) should be deleted from K. It is also correct if all the following states for given state P' have already been obtained in K^{start} (K^{finish}).

(R2.1). State P' is common for K^{start} and K^{finish} if it corresponds to the state $P'' \in K^{start}$ and to the state $P''' \in K^{finish}$ such that there are no elements $P_i'' \in P''$ and $P_i''' \in P'''$ for the same index "i" and such that either $P_1'' = 0$ & $P_i''' = 1$ or $P_1'' = 1$ & $P_i''' = 0$.

0.4.2.3. Making planning sequence in the system with post-conditions

Let us now suppose, that every operator is represented by a pair $< U_i ,\beta_i >$ where β_i is a post-condition which may be interpreted as a goal of executing operator U_i . We will represent β_i by means of some (logical) function g_i .With every state P^{t+1} of a problem we connect a goal set $G^{t+1} = <g_1,g_2,.,g_v>$ with logical functions g_i , i=1,v .

Let P^{t+1} be a current state with the goal set G^{t+1} . The task to be solved is in finding a valid state P^t and a corresponding set G^t to satisfy the following conditions

(i) U_i does not distort P^{t+1} ;

(ii) $G^{t+1} \vdash g_i$ (g_i is a derivation from G^{t+1});

(iii) $G^t \subset G^{t+1}$, $\neg$ ($G^t \vdash g_i$) and G^t is a maximum size set with the above shown properties. We confine our considerations to the supposition that $G^{in} = \varnothing$.

Consider the formula $\neg(G^t \vdash g_i^*)$. According to the Gödel theorem it follows that there is an interpretation I providing truth of every formula in G^t and false value of g_i^*. Suppose,

$$G^{t+1} = \{g_1, g_2, \ldots, g_v\}.$$

Consider now the equation

$$g_k \& \overline{g}_i^* = 1.$$

Its solution may be interpreted by means of a set of intervals, for example, if

$$g_k = P_1 \& P_2 \vee \overline{P}_3$$
$$g_i^* = P_2 \& \overline{P_3}$$

then $g_k \& \overline{g}_i^* = (P_1 \& P_2 \vee \overline{P}3) \& (\overline{P}2 \vee P_3) = \overline{P}2 \& \overline{P_3} \vee P_1 \& P_2 \& P_3$ which gives the following interval representation:

$\quad\quad P_1\ P_2\ P_3$

$I_a = <1 \quad 1 \quad 1>$

$I_b = <* \quad 0 \quad 0>$.

We need some intermediate operations for finding the common part of two intervals. Thus, if $I_1 = < 1 * 0 >$ and $I_2 = < * * 0>$ then $I_{1,2} = I_1 \cap I_2 = <1 * 0>$. By analogy with set difference we find

$$I_{1/2} = I_1 \setminus I_{1,2} = \varnothing$$
$$I_{2/1} = I_2 \setminus I_{1,2} = < 0 * 0 >$$

where \ is a symbol of subtraction operation.

Subtracting one interval from another we may obtain a disjunction of intervals, e.g. if

$$I_A = < * 0 * * 0 >$$
$$I_B = < 1 0 * 0 0 >$$

then

$$I_A \setminus I_B = < 0 0 * * 0 > < * 0 * 1 0 >$$
$$I_B \setminus I_A = \varnothing.$$

Evidently, if $I_A \supset I_B$ then $I_B \setminus I_A = \varnothing$. Suppose further, that for G^{t+1} and g_i^* we subsequently find the individual solutions of equations:

$$g_1 \& \overline{g}_i^* = 1$$

in the form $H_1 = \{I_{11}, I_{12}, .., I_{1Z}\}$

$$g_2 \& \overline{g}_i^* = 1$$

in the form $H_2 = \{I_{21}, I_{22}, .., I_{2Z}\}$ and

$$g_v \& \overline{g}_i^* = 1$$

in the form $H_{1v} = \{I_{v1}, I_{v2}, .., I_{vZ}\}$

where H_j represents disjunctions of all the intervals suiting to equation $g_j \& \overline{g}_i^* = 1$

Let, for instance, $I_X \in H_A$, $I_Y \in H_B$ and $I_X \cap I_Y \neq \varnothing$.

Include in H_A $I_{X,Y}$ and $I_{X\setminus Y}$. Include in H_B $I_{X,Y}$ and $I_{Y/X}$, where

$$I_{x,y} = I_x \cap I_y$$
$$I_{x\setminus y} = I_x / I_{x,y}$$
$$I_{y\setminus x} = I_y / I_{x,y}. \qquad (*)$$

And in general, let us include in H_A all the nonempty intervals produced on the basis of (*) for an arbitrary I_X and every possible interval I_Y from H_μ ($\mu \neq A$). Extending every set H_μ is possible until there are no two similar intervals.

Example.

Let

$$H_1 = \{I_1 =< 0*1* >, I_2 =< *1*1 >\}$$
$$H_2 = \{I_3 =< *00* >, I_4 =< **01 >\}$$
$$H_3 = \{I_5 =< 0**1 >\}.$$

Find

$$I_1 \cap I_3 = \varnothing$$
$$I_1 \cap I_4 = \varnothing$$
$$I_1 \cap I_5 \neq \varnothing.$$

Thus,

$$I_{1,5} =< 0*11 >$$
$$I_{1\backslash 5} =< 0*10 >$$
$$I_{5\backslash 1} =< 0*01 >.$$

Include $I_{1,5}$ and $I_{1\backslash 5}$ in H_1 instead of I_1 .

Include $I_{1,5}$ and $I_{5\backslash 1}$ in H_3 instead of I_5 .

Changing indices of intervals, we obtain

$$H_1 = \{I_1 =< 0*11 >, I_2 =< 0*10 >, I_3 =< *1*1 >\}$$
$$H_2 = \{I_4 =< *00* >, I_5 =< **01 >\}$$
$$H_3 = \{I_1 =< 0*11 >, I_6 =< 0*01 >\}.$$

There is one common interval I_1 in H_1 and H_3 now.

Further we find:

$$I_3 \cap I_4 = \varnothing$$
$$I_3 \cap I_5 \neq \varnothing.$$

Hence, $I_3 \cap I_5$= < * 1 0 1 >

and $I_{3,5}$ = < * 1 0 1 >

$I_{3\backslash 5}$ = < * 1 1 1 >

$I_{5\backslash 3}$ I = < * 0 0 1 >.

Extend H_1 and H_2 to:

$$H_1 = \{I_1 =< 0 * 11 >, I_2 =< 0 * 10 >, I_3 =< *101 >, I_4 =< *111 >\}$$
$$H_2 = \{I_5 =< *00* >, I_3 =< *101 >, I_6 =< *001 >\}$$
$$H_3 = \{I_1 =< 0 * 11 >, I_7 =< 0 * 01 >\}.$$

Repeating these necessary actions by analogy and leaving all intermediate results, we obtain finally:

H_1 = { I_1 = < 0 1 1 1 >

I_2 = < 0 0 1 1 >

I_3 = < 0 * 1 0 >

I_4 = < 0 1 0 1 >

I_5 = < 1 1 0 1 >

I_6 = < 1 1 1 1 > }

H_2 = { I_7 = < 0 0 0 1 >

I_8 = < 1 0 0 * >

I_9=< * 0 0 0 >

I_4 = < 0 1 0 1 >

I_5 = < 1 1 0 1 > }

H_3 = { I_1 = < 0 1 1 1 >

I_4 = < 0 1 0 1 >

I_6 = < 1 1 1 1 >

I_7 = < 0 0 0 1 > },

The last step is an extremely simple one: it is necessary to find an (arbitrary) interval which belongs to the maximum number of sets H_j (j = 1,2,3), i.e. I_4 (in the above example). The indices of all the sets H_j containing a given interval (I_4) define the set of functions G^* , for which

$\neg (G^* \vdash g_i^*)$.

Thus, in our example $G^* = \{ g_1, g_2 , g_3 \}$.

Let us prove now that the set of formulae G^*obtained in such a way is unextendable. Let I_m be an interval on which G^* is defined. Any function from G^* is true on I_m and g_i is false. Besides, any function not belonging to G^* is false on I_m as it should be included in G^* on the contrary. (It is clear that including such a function in G^*, i.e. obtaining a new set $G^{**} \supset G^*$ provides $G^* \vdash g_i$).

With reference to the proof of correctness of the algorithm outlined in § 0.4.2.1 it may be shown that the strategy used to select the next operator U_i to include in K warrants obtaining the necessary sequence K providing mapping

$<p^{in} ,\varnothing > \xrightarrow{K} <p^{fi} , G^{fi} >$

if such a selection keeps the inference relations $G^* \vdash g_j$ for all remaining operators U_j, $j \neq i$.

If selecting operator U_i excludes possibility for the other operator U_j to be selected it is necessary to organize solution as a tree-search procedure with backtracking.

0.5. Finding an optimum interpretation

Many difficulties arise when trying to find an optimum (with respect to the formalization (0-2)) solution. From the most common viewpoint there are two such problems, those which enable us to formalize an optimization criterion in terms of the initial problems, and those for which such a criterion is unclear or impossible (as in the case of partially-defined problems, for instance). For example, Euler's equation in variation calculus

$$F_y - \frac{d}{dx} F_{y'} = 0$$

defines the necessary condition for a function

$$I(y(x)) = \int_a^b F(x,y,y')dx$$

to achieve an optimum value on the interval [a,b].

This example points out the criterion formalized in terms of the initial function F(x, y, y'). The great majority of optimization problems use the property

$$\frac{dF(x)}{dx} = 0$$

of the same type (i.e. derivative of given function F(x) is equal to 0) to find a corresponding value of the variable x providing optimum value of F(x).

The problems of the other type do not give such a formal criterion. Here the accent is on determining the process of solution searching. It is essential that this process is strongly connected to the weak methods and heuristic reasoning. There is a number of methods using cutting to obtain on their basis a precise solution. These methods find and cut subsets $\widehat{X}$ of values not containing optimum ones. If X is an initial set of valid values then cutting reduces it to $\breve{X} = X\backslash\widehat{X}$. The effectiveness of such a strategy depends on the rate of decreasing the row

$$\breve{X}_1 \supset \breve{X}_2 \supset \breve{X}_3 \supset \ldots \supset \breve{X}_k \ldots$$

The well-known methods using cutting strategy are: the branches-and-bounds method, the method of ellipsis in linear programming (suggested by Hachiyan), the method of minorants and so on. Let us give an example of the latter procedure with a convex function f(x) for which one seeks the minimum value on the interval [a, b]. Since f(x) is convex then if $x_1 \le x_2$ and $f(x_1) \le f(x_2)$ it gives

$$\begin{cases} x^* = \arg\min\limits_{x \in [a,b]} f(x) \\ x^* \le x_2, \end{cases}$$

where "argmin f(x)" stands for the argument x^* providing minimum value of the function f(x) on the given interval.

Thus, supposing $x_1 = (a + b - \varepsilon)/2$ and $x_2 = (a + b + \varepsilon)/2$ we obtain:

$$(a) f(x_1) \leq f(x_2) \rightarrow x^* \in [a, x_2]$$

$$(b) f(x_1) > f(x_2) \rightarrow x^* \in [x_1, b].$$

It gives us the possibility to reduce the length of the initial interval by at least two times. There are some important questions concerning the problems of the above type.

(Problem 1) Finding relevant operators (actions) and conditions for their applicability. The operator in a broad sense is understood as an action connected with the task concept. This action may be one of the following type:

- assigning a variable with value;
- term substitution;
- making equations;
- expressing one variables by means of others;
- transformating of expressions;
- solving equations;
- choosing alternatives;
- introducting new objects;
- finding a decision plan;
- defining subgoals;
- changing the original problem conditions;
- verifying results obtained (and some others).

(Problem 2) Determining the property W of the solution related to the original optimization criterion F as shown by the schemes below:

(P2.a) F→ W

(P2.b) W → F

(P2.c) W # F

(P2.d) F → $\overline{W}$

(P2.e) $\overline{W}$ → F

(P2.f) $F \rightsquigarrow W$

(P2.g) $W \rightsquigarrow F$

(P2.h) $F, W \vdash V$

(P2.i) $F \rightarrow W_1$; $F \rightarrow \overline{W_1}$

$W \sim (W_1 \vee W_2)$ $W \sim (\overline{W_1} \vee W_2)$

and some others, where

(P2.a) means that W is a particular case of F;

(P2.b) means that F is a particular case of W;

(P2.c) means that W and F are mutually incompatible;

(P2.d & P2.e) are the same as (P2.a & P2.b) but W is negated with negation operator $(\overline{\ })$;

(P2.f) means that F is a likely reason for W;

(P2.h) introduces new property V which logically follows from F and W;

(P2.i) considers F as a particular reason of W.

(Problem 3) Making a planning sequence to solve the problem. There are some interesting techniques in automatic solution synthesis that were considered above and based on the theorem proving methods. However, when speaking of problem solving one in fact has in mind something more profound than "pure" deductive reasoning. It concerns such abilities as

(P3.1) the ability to interpret solution of a problem;

(P3.2) purposeful behavior;

(P3.3) the extension of the formal boundaries of a problem;

(P3.4) the ability of self-learning on patterns and mistakes;

(P3.5) intuition;

(P3.6) the ability to "catch" the internal problem structure (insight);

(P3.7) the ability to generalize;

(P3.8) the capacity to reason with incomplete facts and inconsistent information;

(P3.9) the ability to make hypotheses;

(P3.10) the ability to ask questions.

It is (at least by now) evident that the computer is incapable of competing with the human mind in these questions. So it seems to be rational to create human-machine integrated solving systems providing an appropriate specialization for both sides (i.e. the human should deal with the "informal" part of a problem and machine with the "formal" aspects of a problem).

0.5.1. Task conceptualization

To provide the necessary means for "manipulation" with task representation we should introduce some basic notions of task conceptualization. In the beginning of this chapter we considered the notion of a problem and problem states. Now we need some more detailed considerations to define the general scheme of a human-machine process in searching for a solution. Let us begin with the notion of the task concept.

0.5.2. The notion of a task concept

Task concept is a semantic whole consisting of a domain-concept, conditions-concept, solution-concept and a concept of transformations.

Concept represents a set of interrelated notions. If the notions are primary (i.e. not defined through other notions) then concept is called atomic. Primary notions include the notions of an object, an action, relation, property and a function.

Notions P_1 and P_2 form semantic whole if one of the following is true:

(A) P_1 and P_2 are connected to each other as a relation and one of its arguments;

(B) P_1 and P_2 are connected to each other as an action and its carrier or subject.

(C) P_1 and P_2 are connected to each other as a function and its argument (object and its property).

By induction it follows that a set $P = \{P_1, P_2, \ldots, P_n\}$ of notions represents a semantic whole, if every P_i in P form a semantic whole with, as a minimum $P_j \in P \setminus \{P_i\}$.

The structure of a concept C is an ordered set of C_i where C_i is a semantic whole contained in the semantic whole representing C. The structure of a concept C corresponds to a graph $G(\tilde{C}, \vec{W})$ with a set $\tilde{C}$ of vertices connected by transitive arcs

$(\overrightarrow{C_k, C_m}) \in \bar{W}$ if the semantic whole determining concept C_m is contained in the semantic whole which represents concept C_k. A task concept has the structure shown in Fig. 0.6. The character of this structure makes impossible an unlimited "enclosure" of the various individual tasks in it. Therefore, we confine our considerations to a definite set of discrete optimization problems, for instance to those given in [15 - 27]. Note that Fig. 0.6 does not pretend to be full or an exhaustive taxonomy. To proceed with our analysis, let us consider some examples.

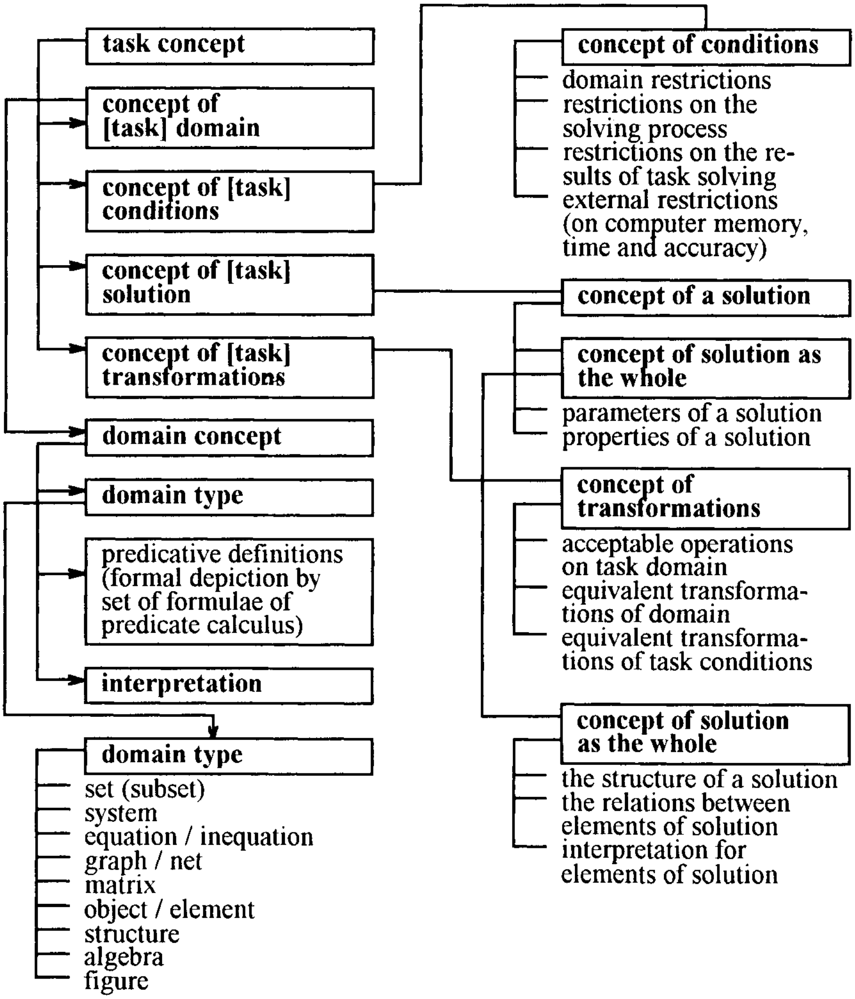

task concept
concept of [task] domain
concept of [task] conditions
concept of [task] solution
concept of [task] transformations
domain concept
domain type
predicative definitions (formal depiction by set of formulae of predicate calculus)
interpretation
domain type
set (subset)
system
equation / inequation
graph / net
matrix
object / element
structure
algebra
figure
concept of conditions
domain restrictions
restrictions on the solving process
restrictions on the results of task solving
external restrictions (on computer memory, time and accuracy)
concept of a solution
concept of solution as the whole
parameters of a solution
properties of a solution
concept of transformations
acceptable operations on task domain
equivalent transformations of domain
equivalent transformations of task conditions
concept of solution as the whole
the structure of a solution
the relations between elements of solution
interpretation for elements of solution

Fig. 0. 6. a)

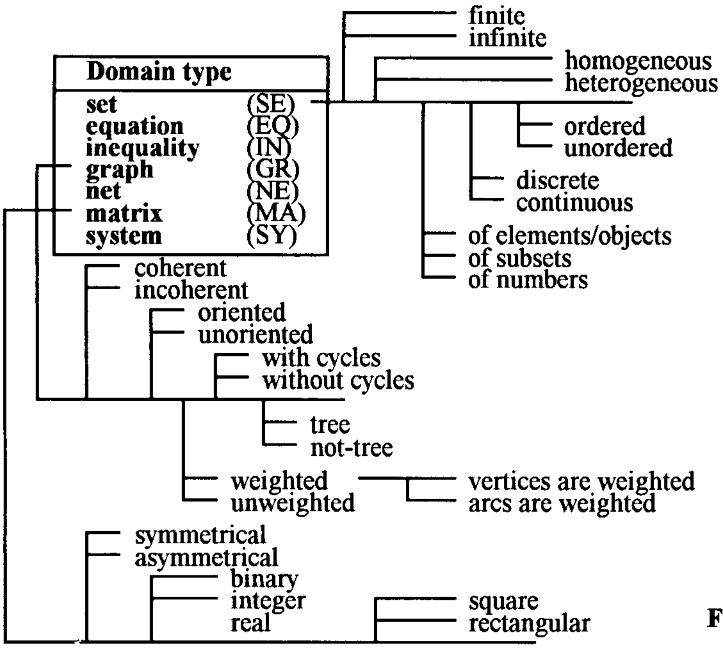


Fig. 0. 6. b)

Concept of conditions

domain restrictions

RD1 restrictions on the sum (or any other function) of elements
RD2 restrictions on the compatibility of elements
RD3 restrictions on the elements precedence(les)
RD4 restrictions on the elements combination(s)
RD5 restrictions on the subordination of elements
RD6 restrictions on the elements belonging
RD7 restrictions on the incompatibility of elements
RD8 restrictions on the values of elements
RD9 restrictions on the availability of elements

restrictions on the solving process

RSP1 sequential choice
RSP2 an influence of step i on the step (i+1)
RSP3 an influence of step i on the step (i+k)
RSP4 no restrictions on the order of including elements in the solution
RSP5 recursive character of a solving process
RSP6 a number of alternatives exponentially rises with the linear increase of the size of task domain
RSP7 including some elements in a solution does not permit one to include others
RSP8 the inclusion some elements in a solution requires to include the others
RSP9 selection of the elements for a solution is performed from a discrete set
RSP10 selection of the elements for solution is performed from a continuous set

Fig 0. 6. c)

Restrictions on the results of task solving

RR1 — a solution represents an unordered set of distinct elements (elements of solution)
RR2 — a solution represents an ordered set of solution elements
RR3 — a solution corresponds to a definite subset of task domain
RR4 — a solution is defined on the whole task domain
RR5 — a solution is presented by the only element from task domain
RR6 — a solution is a structure on the set of elements from task domain
RR7 — a solution is given by the values of the elements [within given groups]

The ralations between elements of solution

ER1 — restrictions on the sum (or any other function) of elements
ER2 — restrictions on the compatibility of elements
ER3 — restrictions on the elements precedence
ER4 — restrictions on the elements combination(s)
ER5 — restrictions on the subordination of elements
ER6 — restrictions on the elements belonging
ER7 — restrictions on the incompatibility of elements
ER8 — restrictions on the values of elements
ER9 — restrictions on the availability of elements

Fig. 0. 6. d)

The structure of a solution

SS1 — matrix (submatrix)
SS2 — (sub)set
SS3 — [required] division into classes
SS4 — sequence (ordered set)
SS5 — list of values for solution elements
SS6 — (sub)graph

Properties of a solution

SP1 — satisfaction of given functional relation
SP2 — optimization of given criterion
SP3 — being the only suitable solution
SP4 — being an arbitrary suitable solution
SP5 — containing the maximum number of elements of solution
SP6 — containing the minimum number of elements of solution

Fig. 0. 6. e)

The examples of task concepts

Let us consider the following concept

< MA (binary, symmetrical) >

< RD2 >

< RSP1, RSP7, RSP9 >

< SS1 >

< ER2 >

< RR3 >

< SP5 >

All of the abbreviations used in this and the following examples are explained in Fig. 0.6b - 0.6e.

This compressed depiction may be interpreted as follows: "There is a binary symmetrical matrix for representing given restrictions on the compatibility of elements from a task domain. It is necessary to determine its submatrix, representing the maximum number of elements in a solution and satisfying the restrictions on the compatibility of elements. The process of finding the elements of a solution is sequential and, moreover inclusion of some elements in the solution may exclude others. The choice of elements is performed from a discrete set". The well-known problem of a maximum clique in a graph [24, 27] entirely suits this situation.

Example 2.

<< SE(finite, homogeneous, of elements)>

< RD3 >

< RSP2, RSP3, RSP9 >

< RR2, RR4 >

< ER3 >

< SS4 >

< SP2 >>

"There is a finite homogeneous set of elements with given precedence relations between elements. The structure of a solution corresponds to ordered sequence of elements providing the optimum value of given criterion. The process of searching for a solution is characterized by the influence of every given step on some future steps (i.e. it is not Markovian)". Such a reading may suit the problem of finding an optimum permutation of given elements from a discrete set with a defined precedence relation [18].

Example 3.

<< SE (of SE(finite, unordered, of elements) >

< RD8 >

< RSP7 >

< RR1, RR7, RR3 ∨ RR4 >

< SP1, SP4 >

< SS5 >>

The problem consists in the following: "there is a set of unordered, finite (sub)sets of elements with given restrictions on their values. It is required to find the values of these elements satisfying a given functional dependency (an arbitrary suitable solution is needed). A final solution is defined (i.e., valid) on the whole domain oron its part".

It is clear that the problem of finding a solution for a given boolean function represented in conjuctive normal form [22] thoroughly corresponds to the given formulation.

Example 4.

<< MA(binary, rectangular, assymetrical)>

< RR3 ∨ RR4 >

< SS1 >

< SP1, SP6 >

< RSP1, RSP3, RSP4, RSP9 >>

"There is given a binary rectangular asymmetrical matrix. It is required to find a (sub)-matrix which satisfies given functional dependency and contains a minimum amount of elements of a solution. The process of solution searching is characterized by dependency between every current and the following steps and consists of choosing elements of solution from discrete sets".

We can see that this formulation is suitable for minimum [cost] cover-problem [17, 27].

0.5.3. Scheme of a solving process

Solving a problem is connected with the purposeful transformation of its concept according to the scheme below which is called S-scheme:

$$C_X \overset{\mu_C}{\rightsquigarrow} C^* \xrightarrow{Rc^*} Q^* \overset{\mu_Q}{\rightsquigarrow} Q_x \tag{0.8}$$

where C_X - is a concept of a task to be solved

C^* - is a concept of a task with a known solution obtained on the basis of an algorithm R_{C^*}

Q^* (Q_X) - is a concept of solution, corresponding to

C^* (C_X)

μ_C , μ_Q - are the concept transformations

The particular cases of (0.8) correspond to the truncated S-schemes (0.9 - 0.11) in which μ_C and μ_Q are autotransformations H_c, H_q that is

$H(C^*) \rightarrow C^*$, $H(Q^*) \rightarrow Q^*$

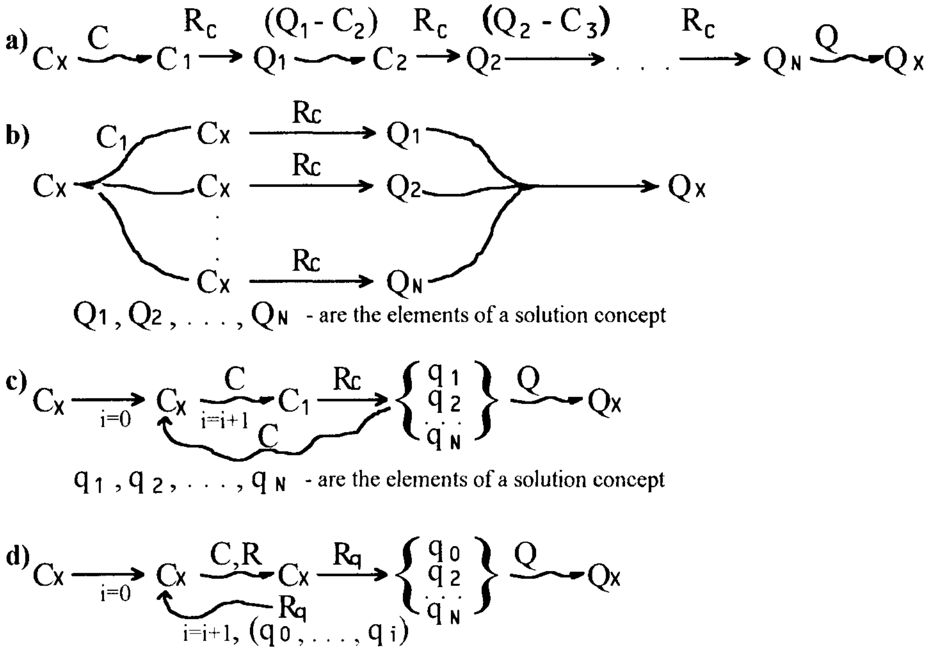
where C^* and Q^* are concepts.

$$C_X \overset{\mu_c}{\rightsquigarrow} C^* \xrightarrow{c} Q^* \text{ (} \mu_Q \text{ - is an autotransformation)} \tag{0.9}$$

$$C_X \xrightarrow{Rc^*} Q^* \overset{\mu_Q}{\rightsquigarrow} Q_x \text{ (} \mu_C \text{ - is an autotransformation)} \tag{0.10}$$

4) $C_x \overset{Rc*}{\text{------>}} Q_x$ (μ_Q ,μ_C are the autotransformations) **(0.11)**

The solving process structures based on S-schemes are shown in Fig. 0.7.



$Q_1, Q_2, \ldots, Q_N$ - are the elements of a solution concept

$q_1, q_2, \ldots, q_N$ - are the elements of a solution concept

a) consequential reduction
b) decomposition
c) simple iteration
d) iteration with finding solution operators R_q

Fig. 0. 7.

The nondeterministic actions during the search for a solution are connected with concept transformations. In general, this action is accomplished by a human-solver in the form of a search tree and it is, therefore, the duty of the computer system to provide automatically all of the necessary manipulations associated with a task concepts. Such manipulations consist of:

(0) checking consistency of the model

(1) addition/deletion of some equation(s)/inequality(ies)

(2) variable substitution

(3) modification of equation(s) / inequality(ies) by means of equivalent transformations

(4) partial calculations

(5) unification of two problem states (contexts)

(6) generation and modification of objects on the basis of rules (productions)

(7) defining a required subset (substructure) on a task domain

(8) substitution of one part of a concept by another

(9) organization of an intelligent helper

(10) testing derivability of one equation (inequality, assertion, etc.) from the others and so on.

0.5.3.0. Checking consistency of the model

Let S be the set of formulae forming a current model. The consistency of S is understood in the sense that S $\vdash \Box$ does not take place. Obviously, if S is consistent and the formula φ is added to S then S ∪ { φ } is consistent if S $\vdash \overline{\varphi}$ does not hold.

0.5.3.1. Variable substitution

Nontriviality of this procedure as well as equation modification may be shown by the following example:

(i) $\int (P'(x) / P(x)) * dif(x)$

(ii) $dif(F(x)) = F'(x) * dif(x)$.

After substitution P for F it should be obtained that

(iii) $\int (dif(P(x)) / P(x)$

and further on the basis of

(iv) $\int \frac{dif(z)}{z} = \ln(z) + C$

it is derived with replacing Z by P(x)

(v) $\ln(P(x)) + C.$

The difficulty is in recognizing the part P'(x) * dif(x) in (i) as a whole because there is term P(x) separating P'(x) from dif(x). Therefore, the system has to use the following rule without which it is impossible to get the final result (v):

$$(A(x) / B(x)) = A(x) * (1 / B(x)) = (1 / B(x)) * A(x).$$

0.5.3.2. Partial calculations and solving equations

Let, for example,

$a = 16$

$b = \sqrt{a^2} + 9$

$c = b + d - 4$ *run.*

Calculations are performed beginning from the row for which run-instruction was applied.

Step 1. Variable "b" is replaced by $\sqrt{a^2} + 9$ what results in

$c = \sqrt{a^2} + 9 + d - 4$

Step 2. Variable "a" is replaced by constant "16":

$c = 21 + d$

Since there is no expression for "d" the variable "c" is assigned with the partially calculated expression above. The special kind of partially calculated expression is an equation (inequality), for example,

$x = x^3 + x^5 - x^{-1} + 1$

or $x > x^2 - 5$

It is therefore essential to recognize the type of functional equation in order to apply a suitable solving method. Partial calculations are connected with a number of technical problems, for example, symbolic arithmetic, making equations, the definition of the equation type and others.

0.5.3.3. Generation / deletion / transformation of task objects on the rule basis

This system function may be demonstrated by the following example. Suppose, there is given an equation:

$$(x_1 \vee \overline{x}_2 \vee x_3)(x_1 \vee x_2 \vee \overline{x}_3)(\overline{x}_1 \vee \overline{x}_3)(\overline{x}_1 \vee x_4 \vee x_3)(\overline{x}_4 \vee \overline{x}_3) = 1.$$

It is desirable that the system would be able to perform the necessary transformations using the rules of the next type:

" if for some i $X_i = 1$ then only those disjunctions which do not contain X_i should remain or which contain $\overline{X}_i$; in this last case delete $\overline{X}_i$ from such disjunctions".

0.5.3.4. Extracting a required subset on the task domain

The system must perform operators

findall (x, P(x)),

findsome (x, P(x)),

where P(x) is a given property of an object belonging to task domain. The above operators are not traditional. The ability to execute these operators depends on the property P(x) as well as on the manner of task domain definition. In the problem solving system, for example, the following operator must be supported

$$\text{findsome} \begin{array}{l}(x, \{x_1 + x_2 - 5x_3 = \max; x_1 - 2x_2 + 6x_3 > 4; \\ 3x_1 + 3x_2 - 3x_3 > 0; x_1 \ge 0; x_2 \ge 0; x_3 \ge 0\}).\end{array}$$

0.5.3.5. Unification of contexts

This problem is one of the most important missions assigned to a problem -solving system. It consists in finding equivalency between two different task models.

0.5.3.6. Orqanization of an intelligent helper

There are some important points concerning human-machine interaction. One of them is the organization of an intelligent helper. The helper is oriented to providing the following types of assistance:

- consulting on the methods or (at least) the ideas of a given problem solution;
- organizing the system of questions and answers in the sense of G. Polya;
- finding an analogous problem in the knowledge-base;
- recognizing the type and the structure of a problem;
- analyzing the possible alternatives when performing decision making;
- retrieving and updating the history of a solution process; and some others,

0.5.3.7. Defining and interpreting the required substructures on a task domain

The system must be able to read and analyze visually displayed information not as the set of symbols but as the definite mathematical structures. For instance, if the following data are displayed

$$b = \begin{matrix} 0101 \\ 1001 \\ 0100 \end{matrix}$$

then the system must recognize that "b" represents an 0,1-matrix. This is the same process employed in the recognition of the variables in an expression. For example, given the expression

$xy + z - x = 2$

then the system has to be clear whether "xy" is a single or a compound variable.

0.6. Psychological aspects

One of the main questions in the development of a human-machine problem solving system concerns mental creativity and productivity. There is a considerable amount of literature related to this issue (see, for example, [28, 29]).

We shall consider the questions partially concerned with this aspect in chapter 6 when discussing heuristic reasoning. However, it should be mentioned that the problem of activating creative processes is a very complicated one and has not been developed sufficiently to be of effective practical utilization.

0.7. Conclusion

It was shown that there are three kinds of general problems in which we are interested, i.e.

(a) - finding interpretations which suit a given task model,represented by the system of relations;

(b) - finding optimum interpretations;

(c) - finding algorithms and some considerations were given to these problems.

Every one of the general problems above may be detailed according to a task model represented by a tuple (0-2). It is evident that the human-machine solving system is essentially based on weak methods mainly oriented to a general problem (c). To develop this approach one needs to consider solving process as the sequence of distinct operators with a priori given properties. It is the subject we shall consider in the following chapters.

Chapter 1

ELEMENTS OF PROBLEM SOLVING THEORY: APPLICATION OF CUTTING STRATEGIES

Abstract. The theory of problem solving operates with a definite set of formal models and corresponding methods. The majority of them deal with a space of potential states of the problem and search for the solution as the path connecting the initial state with the final state of the problem. A short review of these techniques may be found in [15,16].

We confine our considerations to the task consisting in the meeting of the given requirements for a set K of operators. The requirements will be connected with a number of relations which must be satisfied. Moreover, we shall systematically use weak methods and meta-procedures to obtain an exact solution for a problem. This chapter highlights the details of the usage of the converging cuts principle enabling weak methods to become practically strong. Some well-known NP-complete problems are considered for which we apply this principle.

1.1. Introduction

Let us introduce some definitions. We suppose that the search for a problem solution consists in performing a number of elementary solving operations O_i. By elementary solving operation we mean the determination of unknown value(s) on the basis of a known algorithm or rule(s). Therefore, the entire solving process represents an ordered sequence K in the following form K= < O_{i1}, O_{i2} ,....O_{iz} >. Very often, though, it is not obligatory for K to be an ordered set, i.e. the problem consists in finding a set of solving operations with given properties.

Henceforth, we shall consider the notions of an elementary solving operation and an element of a solution to be equivalent. Generally speaking, we do not take into account the computational complexity of the elementary solving operations (e.s.o.) Let us classify different tasks of finding solution K on the basis of the properties of the e.s.o. This classification is provided.

(**i**) *By the very character of the set of e.s.o.:* it may be finite and static, (i.e. be given a priori) or it may be dynamically modified according to known and/or unknown rules.

(ii) *By the character of e.s.o. selection:* the current choice of e.s.o. may have an influence on the future choices or not (i.e. the whole process may be regarded as Markovian or not). In addition, choice of e.s.o. may be performed conditionally or not. In the first case we deal with productions $< C_i, O_i >$, where C_i is a condition restricting the applicability of e.s.o. $O_{i.}$

(iii) *By the availability of the solution definition:* the final state of a problem may be known, i.e. the object we are to obtain is given by its specification, but the algorithm is unknown. On the other hand, the very object may be given through its properties and relations with other objects and in this case we do not know both: the object and an algorithm to obtain it.

(iv) *By the availability of the very solution elements:* there are may be no information concerning the e.s.o. (i.e. they must to be found). In this case the whole problem consists of the following:

(a) finding the e.s.o's ;

(b) finding an algorithm to build the solving sequence of e.s.o's ;

(c) finding the unknown object by application of the algorithm.

It seems that it is this last case that one frequently faces in the applications that are most difficult. It is worthwhile to mention in this context that existing formal approaches to problem solving are scarcely powerful enough for problems of this kind. More realistically, we must consider the whole problem solving process as a tight interaction between human and system supporting that "part" of a search for a solution which may be regarded as formal.

Given that the searching process is clearly connected with finding a set of solution elements, the main task of this chapter is to show regular methods by which this may be accomplished. Such regular methods represent the formalized solving principles.

1.2. The properties of a solution's elements

We distinguish 7 basic properties (relations) of the domain of a solution's elements which are used in the solving process.

Precedence relation ($\succ$). Elementary solving operation O_i precedes the operation O_j ($O_i \succ O_j$) if in every valid solution K O_i is performed before O_j.

Incompatibility relation (#). Two elementary solving operations are incompatible if they cannot be represented in a solution simultaneously (or the choice of one of them excludes the choice of the other, and vice versa). By induction n-ary #-relation means that given combinations of e.s.o. are impossible.

Implication (→). E.s.o. O_i entails e.s.o. O_j (or requires/assumes O_j) if the choice of O_i necessarily leads to the choice of O_j. This is denoted by $O_i \rightarrow O_j$.

Alternative choice (:). We shall say that O_i generates mutually exclusive alternatives and write $O_i \rightarrow O_j : O_k$ if choice of O_i leads to the choice of O_j or O_k which are mutually exclusive.

There is an equivalency between

$$O_i \rightarrow O_j \ : O_k$$

and

$$(\overline{O_i} \vee \overline{O_j} \vee \overline{O_k}) \& (\overline{O_i} \vee O_j \vee O_k).$$

Equivalency (~). Elementary solving operation O_i is equivalent to the sequence $K_i = < O_{j1}, O_{j2}, \dots, O_{jt} >$ of e.s.o. if the results of O_i and K_i are similiar in some context.

Prohibition (–o). Elementary solving operation O_i prohibits O_j (O_i –o O_j) if the choice of O_i excludes the choice of O_j but an opposite may be false (i.e., it is not a symmetrical relation).

Domination ($\triangleright$). O_i dominates O_j in the sense of some criterion (function) φ (what is denoted by $O_i \underset{\varphi}{\triangleright} O_j$) if $\varphi(O_i) \geq \varphi(O_j)$, where φ (.) - is a criterion which is to be optimized (maximized).

We additionally consider the property of existentionality($\exists$) and universality ($\forall$).

Existentionality ($\exists$). The designation $\underset{\varphi}{\exists}$ ($O_{i1}, O_{i2},...,O_{it}$) means that in a given set of e.s.o., there may exist at least one solution element which belongs to the searched sequence K (has a property φ).

Universality ($\forall$). The designation $\underset{\varphi}{\forall}$ ($O_{i1}, O_{i2},...,O_{it}$) means that all the operations from a given set belong to the searched sequence K (have the property φ).

Many problems may be formalized in terms of the given relations. Let us consider them in brief. The basic problems connected to the precedence relation are those belonging to scheduling theory [18]. Since many of them are NP-complete problems they represent an interesting field for the use of weak methods and heuristic reasoning. Relatively few of these problems may be formalized by means of "pure" logic. Some particular cases of the type are considered in this chapter. The main considerations are presented in chapter 3.

Incompatibility (#) - relation is given a central part of the chapter. As will be demonstrated below, a great majority of NP-complete problems may be reduced to a formal representation using the #- relation. Using some equivalent transformations it is also possible to represent other logic relations such as implication ($\rightarrow$), inference ($\vdash$), and prohibition ($-\!\circ$) by means of #. The first part of the chapter is devoted to the interpretation of resolution strategy in a formal logical system on the basis of # - equations.

An equivalency ($\sim$) is a central component of the theory of formal grammars [19]. It, however, is beyond the scope of this book.

Domination ($\triangleright$) is also the subject of discussion in chapter 3.

It is important to note that to find a solving sequence K of the above type all relevant operators O_i must be given. Moreover, we assume that the set of e.s.o is discrete and finite. It is clear that such a simplification in some cases may be not correct. So, we have sufficient reason to separate four kinds of problems of very different natures:

(**i**) the problems with a discrete and finite set of e.s.o;

(**ii**) the problems with a discrete and countable infinite set of e.s.o ;

(**iii**) the problems with a continuum set of e.s.o ;

(**iv**) the problems with an unknown set of e.s.o

This chapter deals only with the problems of type (i). The problems of type (ii, iii) are considered in the next chapter. The problems of type (iv) are discussed in chapter 3.

1.3. A system of axioms for incompatibility calculus

We begin our discussion with a short guide to incompatibility calculus (based on the Sheffer's operation [20]). This calculus is essential for our approach due to its direct applicability to the different well-known NP-complete tasks (as a maximum clique problem and others). It is worth mentioning that reasoning in terms such as "A excludes B, and vice versa" is quite common together with the widely used production form "A implies B". There is a simple equivalency between these two forms of reasoning, namely

$$(A \to B) \sim \left(A \# \overline{B}\right) \qquad \textbf{(1.1)}$$

with symbols → for implication;

# for incompatibility;

‾ for negation;

~ for equivalency.

To build incompatibility calculus it is sufficient to use only this operation (#). However, together with "#" we shall use the negation symbol also for the sake of readability.

Firstly we build a formal system of #-calculus. The axioms will be represented in the form

$$\vdash\frac{A}{B}\dashv$$

where A is an initial set of formulae, and B represents a formula which follows from A due to inference rules of predicate calculus [7,9]. We avoid many details connected with specification of the language, well-formed formulae and inference rules of predicate calculus supposing the reader to be familiar with it. To distinguish different formulae in A we shall write A_1; A_2;...;A_k with designation A_i ($i=\overline{1,k}$) for every individual formula.

The list of axioms is the following

$$\text{(A0)}\ \vdash\frac{x\#y}{y\#x}\dashv;\ \vdash\frac{x\#x}{\bar{x}}\dashv;\ \vdash\frac{\overline{x\#y}}{\begin{array}{c}x\\ y\end{array}}\dashv;\ \vdash\frac{\#x}{\bar{x}}\dashv;$$

$$\text{(A1)}\ \vdash\frac{(x\#y)\#F;\ \bar{x}}{\bar{F}}\dashv;\ \vdash\frac{(x\#y)\#F;\ \bar{y}}{\bar{F}}\dashv;$$

$$\text{(A2)}\ \vdash\frac{(x\#y)\#F;\ F}{\begin{array}{c}x\\ y\end{array}}\dashv;\ \vdash\frac{(x\#y)\#z;}{\begin{array}{c}(z\#\bar{x})\\ (z\#\bar{y})\end{array}}\dashv;$$

(A3) $\vdash \dfrac{(x\#y)\#F;\ x\ \#\ F}{\overline{F}} \dashv$;

(A4) $\vdash \dfrac{(x\#y)\#F;\ x}{\overline{y}\ \#\ F} \dashv\ ;\ \vdash \dfrac{(x\#y)\#F;\ y}{\overline{x}\ \#\ F} \dashv$;

(A5) a) $\vdash \dfrac{(x\#y)\ \#\left(x\#\overline{y}\right)}{x} \dashv$;

b) $\vdash \dfrac{(x\#y)\ \#\left(\overline{x}\#\overline{y}\right)}{\left(x\#\overline{y}\right)} \dashv$;

$$\left(\overline{x}\#y\right)$$

c) $\vdash \dfrac{(x\#y)\ ;\ \left(x\#\overline{y}\right)}{\overline{x}} \dashv$; (resolution rule)

(A6) $(x\#y)\ \#\ \square$ is always true ($\square$ stands for falsehood)

(A7) $\vdash \dfrac{(\ x\ \#\square\ )\ \#\ y}{\overline{y}} \dashv$;

$$\text{(A8)}\ \vdash\frac{\begin{array}{c}(x\#y)\#F;\\ (x\#\overline{y})\#F;\end{array}}{\overline{F}}\dashv\;;\;\vdash\frac{\begin{array}{c}(x\#y)\#F;\\ (z\#\overline{y})\#F;\end{array}}{\overline{F}}\dashv\;;$$

$$\text{(A9)}\ \vdash\frac{(x\#y)\#\overline{x}}{x}\dashv\;;\;\vdash\frac{(x\#y)\#x}{x\#\overline{y}}\dashv\;;$$

$$\text{(A10)}\ \vdash\frac{(\overline{x}\#y)\#F;\overline{x\#y}}{\overline{F}}\dashv\;;\;\vdash\frac{(x\#\overline{y})\#F;\overline{x\#y}}{\overline{F}}\dashv\;;$$

$$\text{(A11)}\ \vdash\!\!-\!\!+\frac{\begin{array}{c}(x\#y)\#F;\\ (x\#y)\#G;\end{array}}{(x\#y)\#(\overline{F}\#\overline{G})}\dashv\;;\;\vdash\frac{(x\#y)\#(F\#G)}{\begin{array}{c}(x\#y)\#\overline{F}\\ (x\#y)\#\overline{G}\end{array}}\dashv\;;$$

$$\text{(A12)}\ \vdash\frac{\#(\;\square\;)}{\text{is always true}\quad;}\dashv\;;\;\vdash\frac{\overline{\#(x,y,u,\dots,w)}}{x,y,\dots,w}\dashv\;;$$

$$\text{(A13)}\ \vdash\frac{(x\#y)\#(z\#\bar{y})}{(\bar{x}\#\bar{z})}\dashv;$$

$$\text{(A14)}\ \vdash\frac{\overline{(x\#y)}\#\overline{(u\#v)}}{\#(x,y,u,w)}\dashv;$$

$$\text{(A15)}\ \vdash\frac{\#(a,b,c,d,\ldots,w);a}{\#(b,c,d,\ldots,w)}\dashv;\ \text{(contraction rule)}$$

$$\text{(A15')}\ \vdash\frac{\#(a,b,c,d,\ldots,v,w);\#(a,b,c,\ldots,v)}{\#(a,b,c,d,\ldots,v,w)\ \text{is to be deleted}}\dashv;$$

All these axioms may be easily verified by equivalence $(a\#b)\sim\bar{a}\vee\bar{b}$. Let us show that the following system is contradictory:

(F1) $\#(a,b,y)$

(F2) $(x\#y)\#a$

(F3) $(b\#y)\#z$

(F4) $x\#a$

(F5) $\bar{z}\#\bar{a}$

(F6) $(\overline{b\#y})\#\bar{a}$.

From (F2) and (F4) on the basis of axiom (A3) we obtain (F7):

(F7) $\bar{a}$.

From (F6) and (F7) we obtain (F8): (b # y). With (F3) it gives (F9): $\bar{Z}$. Finally, from (F5) it follows that (F10): a. Refering to (F7) we derive a contradiction.

To simplify manipulations it is useful to represent every formula of incompatibility calculus as the conjunction of the formulae in the following form

$$\#\left(x_{i1}, x_{i2}, \ldots, x_{ir1}\right)$$
$$\#\left(x_{j1}, x_{j2}, \ldots, x_{jr2}\right)$$

where $x_{i\,h}$ (x_{jm}) denotes some variable or its negation. This is always possible due to the known theorem about representing propositional formulae in the conjunctive normal form. This is also required by resolution strategy on the basis of Robinson's method [21].

1.3.1. Resolution strategy for incompatibility calculus

Agreement on designations:

# - is the symbol of incompatibility relation;

x # y - means that two objects x and y cannot be presented for solution simultaneously;

#(x,y) - is the same as x # y;

#(a, b, c,...,w) - denotes an expansion of the previous case for the set of objects {a, b,c,...,w};

∨,¬ , &,→ ,~ - disjunction, negation, conjunction, implication and equivalency correspondingly.

□ - is the symbol of logical contradiction;

⊢ - syntactic entailment;

|= - semantic entailment (⊢ and |= are equivalent according to Gödel's completeness theorem);

∅ - an empty set;

$\in$- is used to denote the belonging of some object(s) to the given set;

$\cap$ - is the sign of set intersection;

$\cup$ - is the sign of set combining.

We begin our considerations by solving systems of equations representing #-relations between different pairs of objects. Afterwards, we will make generalization from the resolution strategy to a common case. Let us consider the system of equations in the form a # b (objects "a" and "b" are mutually exclusive). An acceptable solution S for such a system represents a set of objects satisfying every equation of the given system. The solution S has the property that either object "a" or its negation "$\bar{a}$" must be included in S. If a $\in$ S then it simply indicates that $\bar{a} \notin$ S. Obviously, a # $\bar{a}$ for every "a". As a sequence, in the case when

$a_i \in S$ and $\bar{a}_i \in S$ or $a_i \notin S$ and $\bar{a}_i \notin S$

the solution S must be denied and is called unacceptable (invalid).

Let us have the system

$$U: \quad \begin{array}{lllll} x_1 \# x_3 & x_2 \# x_4 & x_3 \# \overline{x_5} & x_4 \# x_7 & \overline{x_5} \# \overline{x_6} \\ \overline{x_1} \# x_4 & \overline{x_2} \# x_5 & x_3 \# \overline{x_6} & \overline{x_4} \# \overline{x_7} & \overline{x_5} \# \overline{x_7} \\ x_1 \# x_6 & \overline{x_2} \# \overline{x_7} & & \overline{x_4} \# \overline{x_5} & \\ x_1 \# x_7 & \overline{x_2} \# \overline{x_3} & & & \end{array} \qquad (1.2)$$

All the equations of the form $x_i \# \bar{x}_i$ ($i=\overline{1,7}$) are assumed by default. We are looking for the valid solution S if it exists.

In this section we obtain inference rules for # - calculus directly from the system U of 2-ary equations, although there are more effective procedures to find a solution in this case (these procedures are considered in the next section). Unfortunately, there are no efficient algorithms (from the computational complexity viewpoint) in the general case with n-ary #-equations (n > 2). Therefore, our considerations of inference rules remain valuable from this viewpoint. We directly begin with

Inference rule 1. If simultaneously $x_i \# x_j$ and $\bar{x}_i \# x_j$ for some $i \neq j$, then $\bar{x}_j \in S$.

Proof. By assuming opposite (i.e. $x_j \in S$) we derive $\bar{x}_i \in S$ and $x_i \in S$ and thus, the invalidity of S.

Inference rule 2. If simultaneously $x_j \# x_i$ and $x_k \# \bar{x}_i$ (for some i, j, k) then $x_j \# x_k$.

Proof. If we suggest that $x_j \in S$ and $x_k \in S$ then it entails $\bar{x}_i \in S$ and $x_i \in S$ and invalidity of S.

This inference rule enables one to generate new formulae. For example, from **(1.2)** we consider the formulae with x_1 and $\bar{x}_1$ and obtain :

$$x_3 \# x_4, x_4 \# x_6, x_4 \# x_7.$$

The inference rule 2 also remains operational in the common case concerning relations in the form $\#(a,b,...,c)$.

So, from $\#\left(\overline{x_1}, x_2, \overline{x_3}\right)$

and $\#\left(x_1, x_2, x_4, x_7\right)$

one can derive $\#\left(x_2, \overline{x_3}, x_4, x_7\right)$.

A proof may be simply obtained from equivalence $\#(a,b,\ldots,c) \sim \bar{a} \vee \bar{b} \vee \ldots \vee \bar{c}.$ Also note that it immediately follows from Robinson's principle of resolution [21].

Lemma 1.1. (i) If simultaneously for some i $x_i \in S$ and $\bar{x}_i \in S$ or (ii) $x_i \notin S$ and $\bar{x}_i \notin S$ then S is invalid. If S is invalid due to inference rules, then system U is unresolvable.

Lemma 1.2. If we derive $x_i \in S$ then it also means that $x_j \notin S$ for every x_j such as $x_i \# x_j$.

If there are for some x_i only equations with x_i (not with $\bar{x}_i$) or only equations with $\bar{x}_i$ (not with x_i) then such equations should be deleted from U without loss of solution.

Definition. Two systems U and U' of #-equations are equivalent if any solution of one of them is a solution for the other (or may be transformed to that solution by means of a known algorithm).

Definition. For two equations

$$\#\left(x_i, F\right)$$
$$\#\left(\bar{x}_i, G\right)$$

we call $\#(F,G)$ their x_i -resolvent.(This definition is essential for Chapter 4.)

The solving procedure considers variables in the increasing order of their indices. Thus, firstly we consider all the equations containing the variable $x_1(\bar{x}_1)$ and find all possible x_1 -resolvents which are automatically added to the initial system U excepting the tautologies (i.e., those of the form $x_\alpha \# \bar{x}_\alpha$). Then we remove from U all of the equations with x_1 ($\bar{x}_1$) and repeat our considerations with x_2 ($\bar{x}_2$) in a similar way, etc.

We formalize this process as

Theorem 1.1. (Cutting). Let ψ (x_i) be a set of equations from U containing x_i or $\bar{x}_i$. Then if x -resolvents from ψ (x_i) are added to U one can remove ψ (x_i) from U and preserve equivalency of this new system and U.

Proof. Denote by U(i) the new system of equations $\{U \cup T(i)\} \setminus \psi$ (x_i), where T(i) is the set of consequences from ψ (x_i) obtained by the x - resolutions. We must prove that any valid solution S_i for U(i) does not contradict ψ (x_i). Suppose an opposite, i.e. S_i is in contradiction with ψ (x_i). It means that there is a pair of equations

$x_\alpha \# x_i$ and $x_\beta \# \bar{x}_i$

with x_α ,$x_\beta \in$ S. But it is impossible since we derive from ψ (x_i) $x_\alpha \# x_\beta$ which immediately leads to a contradiction. Therefore, U and U(i) are equivalent in the above defined sense.

An analogous considerations may be applied to any other variable from U(i), etc. until the system of equations U becomes unreducible.

Corollary. If $x_i \in$ S (or $\bar{x}_i \in$S) then having got from this necessary results one can remove all of the equations with x_i ($\bar{x}_i$) from consideration.

Proof. Let $\bar{x}_i \in S$ and all necessary consequences from this be obtained. Consider the formula $x_i \# a$. It is clear that in both cases $a \in S$ or $a \notin S$ ($\bar{a} \in S$) this formula is always true for $\bar{x}_i \in S$, i.e. the choice of "a" does not depend upon a given formula.

Return to the example (**1.2**). After generating $T(x_1)$ and removing $\psi(x_1)$ we get the following configuration

$$\begin{array}{llll}
x_2 \# x_4 & x_3 \# \overline{x_5} & x_4 \# x_7 & \overline{x_5} \# \overline{x_6} \\
\overline{x_2} \# x_5 & x_3 \# \overline{x_6} & \overline{x_4} \# \overline{x_7} & \overline{x_5} \# \overline{x_7} \\
\overline{x_2} \# \overline{x_7} & x_3 \# x_4 & \overline{x_4} \# \overline{x_5} & \\
\overline{x_2} \# \overline{x_3} & & x_4 \# x_6 &
\end{array} \tag{1. 3}$$

From $\psi(x_2)$ we obtain $T_2 = x_4 \# x_5, x_4 \# \bar{x}_7, x_4 \# \bar{x}_3$ and a new configuration

$$\begin{array}{lll}
x_3 \# \overline{x_5} & x_4 \# x_7 & \overline{x_5} \# \overline{x_6} \\
x_3 \# \overline{x_6} & x_4 \# \overline{x_7} & \overline{x_5} \# \overline{x_7} \\
x_3 \# x_4 & \overline{x_4} \# \overline{x_7} & \\
\overline{x_3} \# x_4 & \overline{x_4} \# \overline{x_5} & \\
 & x_4 \# x_6 & \\
 & x_4 \# x_5 &
\end{array} \tag{1. 4}$$

From $\psi(x_3)$ we find $\bar{x}_4 \in S$ and further, due to lemma 1.2, $\mathbf{x_5} \in S$, $\mathbf{x_7} \in S$. Additionally we get new formulae

$$x_4 \# \overline{x_6}, \quad x_4 \# \overline{x_5}.$$

According to the corollary above we exclude the equations with x_4, x_5, x_7 ($\bar{x}_4$, $\bar{x}_5$, $\bar{x}_7$) which results in an empty final configuration with a partial solution

$$\overline{x_1}, x_2, \overline{x_4}, x_5, x_7 \in S.$$

The values of x_3 and x_6 may be obtained by the following common rule:

"if the variables x_{i1}, x_{i2},...,x_{it} remain unknown ($i_t > i_{t-1} > \ldots > i_1$) then their determinantion is obtained by including x_i or $\overline{x}_i$ in S in accordance with lemma 1.2 starting from the variables with larger indices. This process cannot result in a logical contradiction (□)".

Thus, in our case we have

a) $\overline{x_6} \in S \to \overline{x_3} \in S$

b) $x_6 \in S \to x_3 \in S$ *or* $\overline{x}_3 \in S$.

Generalization of the results obtained for n-ary (n > 2) #-relations may be performed without difficulty.

So, inference rule 1 is replaced by

Inference rule 1*.

(i) if simultaneously

$$\#\left(a, b, c, \ldots, w, x_j\right)$$
$$\#\left(\overline{a}, b, c, \ldots, w, x_j\right)$$
$$\#\left(a, \overline{b}, c, \ldots, w, x_j\right)$$
$$\cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot$$
$$\#\left(\overline{a}, \overline{b}, \overline{c}, \ldots, \overline{w}, x_j\right)$$

then $\overline{x_j} \in S$

(ii) if simultaneously

$$\#\left(x_i, R\right)$$
$$\#\left(\overline{x_i}, R\right) \quad \text{where } R \text{ is a set of objects},$$

then $\overline{R} \in S$.

Thus, if $R = \{a, b\}$, then $\overline{R} = \{\overline{a}, b\} \vee \{a, \overline{b}\} \vee \{\overline{a}, \overline{b}\}$

Inference rule 2 is replaced by

Inference rule 2*. If

$$\#(x_i, R)$$
$$\#(\overline{x_i}, Q)$$
then $$\#(R, Q)$$

where R , Q are arbitrary subsets of objects.

Example.

From $\#(a, b, c)$

$\#(\overline{a}, e, f, g, h)$

we obtain $\#(b, c, e, f, g, h)$

Corollary.

From $\#(a, F \vee G)$

and $\#(\overline{a}, H \vee E)$

we derive

$\#(F,H)$

$\#(F,E)$

$\#(G,H)$

$\#(G,E)$

Inference rule 3 retains its formulation. In addition we introduce the rule of simplification, namely :"if there are equations in the form # (F,G) and # (G) then the first one may be deleted without loss of solution". The proof immediately follows from the observation that every solution of # (G) is also suitable for # (F,G).

On the basis of a given resolution strategy we now produce more efficient algorithms.

1.3.2. The case of 2-ary #-equations

For the systems with 2-ary (binary) #-equations there is a simple and efficient algorithm based on a binary-tree equivalent procedure [22].

Due to the very form of 2-ary #-equations it becomes practically effective to try alternative substitutions for variables i.e. to exercise both variants x = 1 and x = 0.

If we cannot derive a contradiction from a substitution for a variable x, then we can reduce the initial system by deleting all of the formulae with x (or $\bar{x}$).

So, for the system (**1.2**) supposing $x_4 \in S$ we derive a contradiction, therefore, it is deduced that $\bar{x}_4 \in S$ and now we get $\bar{x}_1, x_2, x_5, x_7 \in S$ without contradiction. After reducing U we get the equation $x_3 \# \bar{x}_6$.

This may be summarized by the following :

Lemma 1.3. If setting of $x_1, x_2 ,..., x_k$ does not lead to a contradiction then all the equations containing these variables or their negations may be deleted without loss of a solution.

Proof. If the rest part U^* of U which does not contain the variables $x_1, x_2 ,...,x_k$ is unresolvable then addition to U of equations with $x_1,...,x_k$ ($\bar{x}_1,...,\bar{x}_k$) does not eliminate contradiction.

1.3.3. The case of n-ary #-equations

In this section we will develop a combined method on the basis of resolution strategy and some reasonable heuristic method of searching for "the most" probable way to the goal. Our prime task is not to refute a given system of #-equations but to solve the system with some valid solution. Let us consider the system (1.5).

Let us answer the question: "what is the estimation of the mathematical expectation of a number of valid solutions for *U*?". To answer the question denote by V_φ the number of arguments of a formula φ. The number of interpretations for φ is 2^{V_φ} and only one is invalid (i.e. providing falsehood of φ). So, we may introduce a probability of φ to be true for every randomly generated interpretation I and denote it by tr(φ).

U:

$$\begin{aligned}
&\#\left(a,b,\bar{c}\right)\\
&\#\left(c,d,\bar{e},\bar{b}\right)\\
&\#\left(\bar{a},\bar{e},\bar{d}\right)\\
&\#\left(d,e,f\right)\\
&\#\left(\bar{d},\bar{f},c\right)\\
&\#\left(b,c\right)\\
&\#\left(a,\bar{e}\right)\\
&\#\left(e,f\right)\\
&\#\left(\bar{a},\bar{f}\right)
\end{aligned} \tag{1. 5}$$

Thus we have

$$tr(\varphi) = \frac{2^{V_\varphi} - 1}{2^{V_\varphi}} \tag{1. 6}$$

Note. For the validity of the whole procedure it is required that the initial system does not contain duplicates of any formula and that all the contractions which may be made by the contraction rule (A15') are performed.

For the entire system *U* we obtain the approximated probability of an arbitrary interpretation to be the valid solution for *U* in the form

$$Sol(U) = tr(\varphi_1) * tr(\varphi_2) * \ldots * tr(\varphi_z) \tag{1. 7}$$

where z is the number of equations in *U*. Note, that **(1.7)** does not take into account the conditional character of the probabilities tr(φ_i). Let M denote the total number of variables in *U*. Therefore 2^M is the total number of different interpretations for *U*. The mathematical expectation of the number of valid solutions for U (denoted by v) may be expressed approximately as

$$v = 2^M * Sol(U) \tag{1. 8}$$

We use a good heuristic with the aim of locating solutions or performing a cutting on the basis of a partial result obtained.

A heuristic results from the following reasoning. At every step we must select into solutions some variable (or its negation). This choice must be drawn from the set of alternatives. We must decide which choice is "the best"? To do it we have to bear in mind that failure of the whole process is connected with an emptiness of such a set. This is due to the fact that all variables from some formula #(x_1 ,..., x_2) have been included in S. Therefore we must adhere to the strategy that prefers initially to consider the "shortest" formulae (with minimum number of arguments). If we choose some variable Z then the system U is to be reduced by deleting all the formulae with $\bar{Z}$ and removing Z from the formulae containing Z. Let us denote by n_2 -the number of 2-ary #-equations, by n_3 - the number of 3-ary #-equations etc. Thus for system (**1.5**) we have $n_2 = 4$, $n_3 = 4$, $n_4 = 1$. After including, for instance, "a" and then deterministically "e", "$\bar{f}$" in S, U will be reduced to

$$U^{(a)} : \begin{array}{l} \#(b,\bar{c}) \\ \#(\bar{d},c) \\ \#(b,c) \end{array} \tag{1. 9}$$

with $n_2' = 3$.

An "effect" of appointing variable "a" to S may be estimated from the value of Sol($U^{(a)}$)=0.42 on the basis of formula (**1.7**). Thus, we get

$Sol(U^{(a)}) = 0.42$

with $v_a = 3.3$

From the considerations above it follows that we should prefer such a choice which provides a maximum estimation of the value v' for reduced system U' due to appointment given variable to S.

Estimating these values for variables of the system (**1.5**),we find that either variable "$\bar{b}$ " or variable "$\bar{c}$ " should be included in S in the first instance.

Suppose $\bar{b} \in$ S. Reduced system U has the form

$$U'\!: \quad \begin{array}{l} \#\left(c,d,\bar{e}\right) \\ \#\left(\bar{a},\bar{e},\bar{d}\right) \\ \#\left(d,e,f\right) \\ \#\left(\bar{d},\bar{f},c\right) \\ \#\left(a,\bar{e}\right) \\ \#\left(e,f\right) \\ \#\left(\bar{a},\bar{f}\right) \end{array} \tag{1.10}$$

Then "$\bar{c}$" may be included unconditionally (because there is no equation with "$\bar{c}$"). This time we obtain

$$U'': \begin{array}{l} \#\left(\bar{a},\bar{e},\bar{d}\right) \\ \#\left(d,e,f\right) \\ \#\left(a,\bar{e}\right) \\ \#\left(e,f\right) \\ \#\left(\bar{a},\bar{f}\right) \end{array} \tag{1. 11}$$

with final solution $\bar{b}$, $\bar{c}$, a, e, $\bar{f}$ $\in$ S

It is curious to note that this strategy for the system of 2-ary equations prefers to choose those variables which reduce the number of equations as much as possible.

All that remains is to show how to perform cutting in the case of algorithm failure. Let us suppose that a set of appointments has the form

$$S' = \{x_{i1}, x_{i2}, \ldots, x_{it}\}$$

with last appointment x $_{it}$ produced by the algorithm. S' has a property that we cannot add to it any other variable (or its negation) from the remaining system without obtaining a contradiction.

To proceed we must introduce the following counterpart of the theorem 1.1.

Theorem 1.2. Any set of #-formulae containing variable x and $\bar{x}$ may be equivalently replaced by their resolvents due to the rule of resolution, i.e. from # (x,F) and ($\bar{x}$,G) follows # (F,G).

As we remember, #(F,G) is called x-resolvent of equations # (x,F) and # ($\bar{x}$,G). Theorem 1.2 asserts that the initial system U of #-equations may be replaced by the equivalent system of Z-resolvents for any variable Z ($\bar{Z}$) together with those equations which do not contain variable Z ($\bar{Z}$).

Before proving this theorem let us consider an example. Suppose, we have a system

U:

$$\#(a,c,d)$$
$$\#(a,c,\bar{d})$$
$$\#(\bar{a},\bar{c})$$
$$\#(\bar{a},\bar{d}).$$

Then an equivalent system *U'*, obtained from *U* on the basis of d-resolvents has the next form:

$$U':\begin{array}{l}\#(a,c)\\ \#(\overline{a},\overline{c}).\end{array}$$

Another equivalent system U'' obtained on the basis of a-resolvents has the next form:

$$U'':\#(c,\overline{d}).$$

Note, that tautologies (i.e., #-equations containing pair (s) of contrary arguments x and $\overline{x}$) may be deleted. For example

$$\begin{array}{l}\#(a,c)\\ \#(\overline{a},\overline{c},d)\end{array}$$

give $\#(c,\overline{c},d)$ which is always true.

As far as this example is concerned theorem 1.2 may be understood in such a way that any valid solution of U' or U'' does not contradict U (i.e. may be expanded to the valid solution of U).

Denote the system obtained from U with x-resolvents by U^* (U' or U'' are individual examples of U^*). We must prove that every valid solution for U^* is suitable for U (an opposite is evident due to the rules of logical inference). Suppose $<\alpha, \beta,...,\gamma>$ is a valid solution for U^* and invalid for U. It means that there is a variable $x \notin \{\alpha, \beta ,...,\gamma\}$ such that

$<\alpha, \beta ,...,\gamma> \rightarrow x \in S$

and $<\alpha, \beta ,...,\gamma> \rightarrow \overline{x} \in S$

simultaneously. More precisely, it means that there are two formulae

$$\begin{array}{l}\#(x,Z_1)\\ \#(\overline{x},Z_2)\end{array}$$

with $Z_1, Z_2 \in \{\alpha, \beta,..., \gamma\}$.

The latter directly leads to a contradiction.

Theorem 1.2. gives us the possibility to create an effective cutting mechanism. Return to the partial invalid solution S' = {x_{i1}, x_{i2} ,...,x_{it} }. We came to a conclusion that system $U_{S'}$ cannot be solved. For the sake of clarity let us consider the next example

U:

$$\#(a,b,\bar{c},d,e)$$
$$\#(\bar{a},c,\bar{e})$$
$$\#(\bar{c},\bar{d},\bar{e})$$
$$\#(b,\bar{d},e)$$
$$\#(c,d)$$
$$\#(a,b,\bar{e},\bar{d})$$
$$\#(b,\bar{e},d)$$
$$\#(\bar{b},\bar{d},\bar{c},\bar{e}).$$

With S ' = <a,b> we obtain

$U_{S'}$:

$$\#(\bar{c},d,e) \qquad / f_1 /$$
$$\#(\bar{c},\bar{d},\bar{e}) \qquad / f_2 /$$
$$\#(\bar{d},e) \qquad / f_3 /$$
$$\#(c,d) \qquad / f_4 /$$
$$\#(\bar{e},\bar{d}) \qquad / f_5 /$$
$$\#(\bar{e},d) \qquad / f_6 /$$

$U_{S'}$ is not solvable (by a valid solution). Every equation in $U_{S'}$ has its counterpart in initial system U. Our idea is in the following: to replace U in such a manner that formulae in a new system would possibly contain a combination <a,b>. Because S' is invalid then it follows that #(a,b) and therefore every formula of the type #(a,b,X) with an arbitrary set X may be deleted. To do this, let us find the complementary set S" of the set S':

S" ={c,d,e}

and then reduce subsequently the system U to the system U^{REDUCED} by means of c-, d-, and e-resolutions.

Theorem 1.2. warrants that every valid solution of U^{REDUCED} may be extended to a valid solution of initial system U. If from U^{REDUCED} we derive a contradiction, it means that an initial system U is contradictory. We avoid formal details connected with this procedure. Instead we may summarized with the following semiformal scheme:

s.**i**. Step-by-step select the variables to the partial solution S' until S' becomes invalid or an "end" will be reached.

s.**ii**. If S' is invalid then find the complementary set S".

s.**iii**. Reduce U by generating x-resolvents for variables x not belonging to S". Build new system U.

s.**iv**. Resume procedure for U^{REDUCED}

If "end" then reduce an initial system U on the basis of the solution obtained and repeat the procedure with this nonempty reduced system. Thus, in our example one obtains:

$$U^{REDUCED} = \#\left(a,b\right).$$

Hence, one can set

$\bar{b} \in$ S, $\bar{d} \in$ S

and determine a suitable solution: $\{\bar{c}, e, \bar{b}, \bar{d}\}$.

1.4. An algorithm for searching for maximum-size zero-submatrix

Let $B_{n,n}$ ($n \geq 2$) be a symmetrical 0,1-matrix with zero diagonal. It is required to find a maximum zero submatrix of this matrix with equal set of rows and columns. Thus, for 0,1-matrix in Fig.1.1a the maximum zero-submatrix is shown in Fig.1.1b. So we are interested in an algorithm to solve this problem.

	1	2	3	4	5	6	7	8
1	0	1	1	1	1	0	0	0
2	1	0	1	0	1	0	1	0
3	1	1	0	0	0	1	0	0
4	1	0	0	0	1	1	0	0
5	1	1	0	1	0	0	0	1
6	0	0	1	1	0	0	1	0
7	0	1	0	0	0	1	0	1
8	0	0	0	0	1	0	1	0

Fig. 1. 1. a)

	3	4	7
3	0	0	0
4	0	0	0
7	0	0	0

Fig. 1. 1. b)

$\pi =$

	1	6
1	0	0
2	1	0
3	1	1
4	1	1
5	1	0
6	0	0
7	0	1
8	0	0

Fig 1. 1. c)

The reader may find existing approaches to this problem in [17,23]. An NP-completeness of this problem is shown in [24]. Due to this fact, the approaches based on the use of a binary search tree or on restricted "try-and-choose"-strategy (for example in Geoffrion's algorithm) work poorly with the sparse 0,1-matrices. In particular, approbation of Geoffrion's algorithm on the IBM PC/XT for individual tasks and 0,1-matrices containing 40-60 rows (columns) has shown that the implementation of the procedure took from 1-2 to 150 minutes (in some cases).

Suppose that a rapid heuristic method enables one to obtain a precise solution with probability p (i.e. for N different individual tasks (with relatively large N) a precise solution is obtained in N*p cases). In addition, assume that the algorithm is applied to the same individual task m>1 times with cutting any previously found solution by means of special matrix transformations. Then the value $(1 - p)^m$ represents a probabalistic estimation of not locating the optimal solution. Even with $p = 0.3$ it is sufficient to repeat algorithm $m = 8$ times in order to provide value of $(1 - p)^m$ equal to

0.057. In other words, with a strong heuristic algorithm it is possible to organize a probabilistic estimation of the number of iterations required.

The idea of the suggested approach is in the following.

1. Every iteration results in finding some solution (some zero submatrix). Let its size be equal to K (K represents the number of rows (and columns since the submatrix is square with the same set of rows and columns). Denote by μ (k+1) the mathematical expectation of the number of zero submatrices with (k+1) rows (columns) contained in initial matrix $B_{n,n}$

2. If μ (k+1) < 1, then the procedure stops.

If μ (k+1) $\geq$ 1, then the procedure continues after excluding any solution obtained by special modification of matrix B.

TASK FORMALIZATION

Let $B_{n,n}$ be a symmetrical (in regard to its main diagonal) 0,1-matrix with the dimensions n×n. We will omit, as a rule, the indices of the dimensions. Let $\underset{\sim}{b}$ designate a set of rows (columns) (given by their numbers or indices) of an arbitrary zero submatrix of given matrix B. R ($\underset{\sim}{b}$) will denote the number of elements of $\underset{\sim}{b}$. We may, on occasion, use the designation $\underset{\sim}{b}$ (B) to identify the matrix B we are considering.

We are interested in finding the set $\underset{\sim}{b}^*$ with maximum value R($\underset{\sim}{b}^*$).

The designations R(b^*) and R^* are equivalent; so are the designations R ($\underset{\sim}{b}$) and R.

The designation $\underset{\sim}{b}$ will also be used to identify a matrix with a set of rows (columns) contained in $\underset{\sim}{b}$.

Rows' (columns') numbers will be designated by i, j,..., m, n or by i_1, i_2,...,i_n or by the numbers 1,2,...,N. We shall use Greek symbols α, β,...,γ, with the same aim.

If B(i,j)=1 then we consider rows i and j (columns i and j) to bc incompatible (mutually exclusive), i.e. i # j.

Let there be two rows i and j with i # j. We will say that a row "i" ∅-covers a row "j" if for every column k, k ≠ i and k ≠ j, we have that if B(j,k) = 1 then it follows that B(i,k) = 1 also (the opposite may be false).

Row i and row j (column i and column j) are mutually compatible if the relation i # j does not apply (i.e., B(i,j) = 0).

Define for all compatible rows i and j their disjunction $\underset{\sim}{V}(i,j) = t$, where t represents a new row such that t(k) = B(i,k)∨B(j,k) for every k ≠ i and k ≠ j and B(t,i) = B(t,j) = 0.

For example, $\underset{\sim}{V}(2,4)$ = <10101110> for the rows 2 and 4 from matrix in Fig. 1.1a.

Let $i_1, i_2, \ldots, i_t$ be mutually compatible rows (i.e. there is no pair i_m, i_n with i_m # i_n).

Disjunction $\underset{\sim}{V}(i_1,i_2,\ldots,i_t)$ is defined as follows:

$$\underset{\sim}{V}(i_1,i_2,\ldots,i_t)=\underset{\sim}{V}(i_1,\underset{\sim}{V}(i_2,\underset{\sim}{V}(i_3,\ldots,\underset{\sim}{V}(i_{t-1},i_t)\ldots).$$

Suppose the rows $i_1, i_2, \ldots, i_t$ belong to $\underset{\sim}{b}^*$ (and therefore are mutually compatible).

Let us say that a row α ($i_1, i_2, \ldots, i_t$)- covers a row β (α#β is assumed by default) if a new row α' obtained as a disjunction

$$\alpha' = \underset{\sim}{V}(i_1,i_2,\ldots,i_t,\alpha)$$

∅ – covers a new row β' obtained as a disjunction

$$\beta' = \underset{\sim}{V}(i_1,i_2,\ldots,i_t,\beta).$$

Naturally, the notion we have introduced above assumes compatibility of α (β) with every row from ($i_1, i_2, \ldots, i_t$).

To solve the original problem we will use the following lemmas.

Lemma 1.4. If a row i is zero then it follows that i ∈ $\underset{\sim}{b}^*$

Using lemma 1.4 one may contract the initial matrix $B_{n,n}$.

Lemma 1.5. If a row "i" Z-covers a row "j" (Z may be equal to ∅) then one may exclude row "i" and column "i" from $B_{n,n}$ without losing a solution.

Proof. (for the case of ∅ - covering). Let a row "i" ∅ - cover a row "j". Suppose that $\underset{\sim}{b}^*$ contains "i". It is evident that "i" may be replaced by "j" in $\underset{\sim}{b}^*$ without loss of a solution.

Lemma 1.6. If row "i" contains the only non zero element in column "j" (i.e. all but one element B(i,j) are zero) then one may delete the row "j" and the column "j" without loss of a solution.

Lemma 1.7. Suppose, we have some current solution $\underset{\sim}{b}$ with corresponding value of $R(\underset{\sim}{b})$, where $\underset{\sim}{b}$ determines some zero submatrix of an initial 0,1-matrix B. Then one may remove from B the rows and corresponding columns which contain a number of zeroes less than or equal to $R(\underset{\sim}{b})$. This removal does not lead to the loss of a solution.

Proof. Obviously, every row "i" (column "i") containing a number of zeroes which is less than or equal to a given value n may be compatible at most with n other rows (columns). An alignment $n \le R(\underset{\sim}{b})$ excludes the possibility for such a row "i" to belong to a better solution $\underset{\sim}{b}^*$ with $R(\underset{\sim}{b}^*) > R(\underset{\sim}{b})$.

Corollary. Suppose a row "i" contains n zero elements, $n = R(\underset{\sim}{b})+1$. In order to test that "i" belongs to a better solution $\underset{\sim}{b}'$, one needs to build a submatrix with a set of rows (columns) containing row (column) "i" and rows (columns) s (s = j,k,l,...) with B(i,s)=0. If this submatrix is zero then we obtain a new record $\underset{\sim}{b}'$ such that $R(\underset{\sim}{b}') = R(\underset{\sim}{b}) + 1$. Otherwise, remove the row "i" (column "i") from B.

Lemma 1.8. If every row of 0,1-matrix B contains precisely two units then one may delete from B an arbitrary row (and corresponding column) without loss of a solution.

Proof (a sketch). Any 3 rows may be mutually incompatible. In this case only one of these may be included in solution $\underset{\sim}{b}^{*}$. Let us remove all of such rows from B. For the geometrical interpretation of the remaining set of rows we may use a cyclic chain with the arrows connecting every two incompatible vertices corresponding to the appropriate rows (Fig. 1.2a).

Remove from this chain some vertex i. Then there is a set $\underset{\sim}{b}^{*}$ in the resulting chain (Fig.1.2b) which contains two end vertices due to lemma 1.6: j and k. Moreover, there is no such a set $\underset{\sim}{b}_{1}^{*}$ which does not contain both end vertices j and k and provides $R(\underset{\sim}{b}^{*}) = R(\underset{\sim}{b}_{1}^{*})$. From this, one may conclude that including vertex "i" in $\underset{\sim}{b}_{1}^{*}$ may lead to finding (at the most) some new equivalent solution but not an improving the existing record.

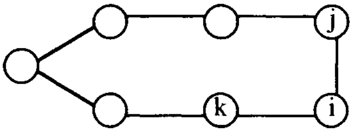


Fig. 1. 2. a)

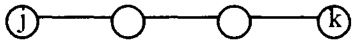


Fig. 1. 2. b)

Lemma 1.9. Let B contain k ≥ 2 similar and mutually compatible rows containing k or less "**1**"**s** in the columns $j_1, j_2,\ldots,j_{l \le k}$. Then one may remove rows $j_1, j_2,\ldots,j_l$ (and columns with the same names) from B without losing the solution.

Proof (for the case k = 2; larger values of k may be automatically reproduced). Let rows α and β be the same (i.e. equal). Assume that each of them contains "**1**"s in the columns j_1 and j_2. If optimum solution is represented by the rows (columns) $\{j_1, j_2) \cup Z$ then there is another optimum solution represented by $\{ \alpha, \beta \} \cup Z$.

METHOD DEFINITION 1

(*The notion of a basic structure*). Let i be the row which may be removed from B without the loss of a solution. We call such a choice (removal) determinate. The cases of determinate choices are defined by the lemmas 1.4 - 1.9 and by their corrolaries.

It is clear that if B0 represents a zero (sub)matrix obtained from B by means of only determinate removals of its rows and corresponding columns, then B0 is a maximum zero (sub)matrix.

As an example, consider the matrix depicted in Fig. 1.1a once more. Step-by-step remove from this matrix the rows and the corresponding columns containing at every step a maximum number of "1"s until matrix becomes zero. Thus, by deleting the rows (columns) i_1, i_6, i_5, i_2, i_8 in the given order, we obtain $\underset{\sim}{b} = \{i_3, i_4, i_7\}$ and $R(\underset{\sim}{b}) = 3$. Using this record and lemma 1.7 we can collapse the initial matrix by means of only determinate removals in the same order. It is therefore shown that $\underset{\sim}{b}^* = \{ i_3, i_4, i_7 \}$.

Every row which may be determinately removed from B is called D-row. A set of D-rows will be designated as a D-set or simply as D.

Assume, that the current matrix B (not D) does not contain any D-row. Then the choice of a row i which is to be deleted is called indeterminate and row i in this case is called an N-row. A set of N-rows will be by analogy designated as an N-set or simply as N. Thus, earlier we selected as N-rows those which contained a maximum number of units at the moment of deleting.

From now on we will use this principle for choosing N-rows. In addition, the D and N-sets have to be ordered in such a manner that row i occupies a position to the left of row j in the same set if row i was selected earlier than row j.

A basic structure represents 3-tuple

$$\langle \;\; N, \;\; D, \;\; \underset{\sim}{b}_{\mathrm{N,D}} \; \rangle \qquad \textbf{(1. 12)}$$

where N, D - are corresponding N, D - sets

$\underset{\sim}{b}_{N,D}$ - is a zero (sub)matrix derived from B by deleting the rows (columns) $N \cup D$.

So, for the problem given in Fig.1.1a we may identify one of its basic structures in the form (1.13):

$$\left\langle\ N=\{i_1,i_6\};\ D=\{i_2,i_5,i_8\};\ \underset{\sim}{b}_{N,D}=\{i_3,i_4,i_7\}\ \right\rangle. \tag{1. 13}$$

It is clear, that if $N = \varnothing$, then the structure $<\varnothing, D, \underset{\sim}{b}_{\varnothing,D}>$ provides $\underset{\sim}{b}_{\varnothing,D} = \underset{\sim}{b}^*$ for given 0,1-matrix B. A matrix B' obtained as a result of adding to a matrix B a new row and column α will be called α-expansion of matrix B and will be denoted as αB.

METHOD DEFINITION 2

(Solving procedure). Let matrix B be given and $D = \varnothing$. As earlier, an N-row is a row with a maximum number of "1"s (at current step of an algorithm). Remove successively N- and D-rows (and corresponding columns) from B to obtain basic structure (1.12). In particular, when an N-set contains two or less, elements an optimal solution $\underset{\sim}{b}^*$ may be found simply as described below. Suppose, $N = \{\alpha,\beta\}$. Designate a set $D \cup \underset{\sim}{b}_{N,D}$ of rows numbers by B_{BASIC}. In α-expansion αB_{BASIC} (in β-expansion β B_{BASIC}) retain only those rows (columns) each of which is compatible with α (correspondingly with β). Thus, we obtain new matrix which will be denoted as $\overline{\alpha\, B_{BASIC}}$ (correspondingly $\overline{\beta\, B_{BASIC}}$).

Lemma 1.10. There exists basic structure for $\overline{\alpha\, B_{BASIC}}$ and $\overline{\beta\, B_{BASIC}}$ of the form $<\varnothing, \overline{D}, \underset{\sim}{b}^*>$.

Proof. Consider the basic structure $<\varnothing, D, \underset{\sim}{b}_{N,D}>$ for B. Keeping in $D \cup \underset{\sim}{b}_{N,D}$ only those rows which are compatible with row α (row β), we obtain a required structure. This trivial property enables us to find $b^*(\alpha B_{BASIC})$ and $b^*(\beta B_{BASIC})$.

Lemma 1.11.

(1) If

$$R\left(b^*\left(\alpha B_{BASIC}\right)\right) \langle R\left(b^*\left(\beta B_{BASIC}\right)\right)$$

then

$$b^*\left(\alpha\beta B_{BASIC}\right) = b^*\left(\beta B_{BASIC}\right)$$

(2) If

$$R\left(b^*\left(\alpha B_{BASIC}\right)\right) = R\left(b^*\left(\beta B_{BASIC}\right)\right)$$

then assuming that

$$R\left(b^*\left(\alpha\beta B_{BASIC}\right)\right) = R\left(b^*\left(\alpha B_{BASIC}\right)\right) + 1$$

we derive that

$$\alpha, \beta \in b^*\left(\alpha\beta B_{BASIC}\right).$$

Proof. Obviously, in the best case $b^*(\alpha\beta B_{BAS\ IC})$ contains as many rows as the best solution from $\{ b^*(\alpha B_{BAS\ IC}), b^*(\beta B_{BAS\ IC})\}$ plus "1". This is possible only when both α and β belong to $b^*(\alpha\beta B_{BAS\ IC})$. For instance, if α belongs to $b^*(\alpha\beta B_{BAS\ IC})$ but β does not belong to $b^*(\alpha\beta B_{BAS\ IC})$ then the condition (2) does not hold:

$$(2)\ R\left(b^*\left(\alpha B_{BASIC}\right)\right) = R\left(b^*\left(\beta B_{BASIC}\right)\right)$$

and vice versa.

Let us start with the general case. It is supposed that the basic structure is given and $N = \{ n_1, n_2, \ldots, n_x \}$, $D = \{ d_1, d_2, \ldots, d_y \}$, $b = \{ b_1, b_2, \ldots, b_z \}$.Create a covering matrix π with a set of rows $\underset{\sim}{b}_{N,D\cup N\cup D}$ and a set of columns N. On the intersection of a row b_i

and a column n_j write "1" if and only if rows b_i and n_j are incompatible in B (i.e. B(b_i, n_j)=1). Thus, for the basic structure (1.13) from the example in Fig. 1.1a we obtain matrix π shown in Fig. 1.1c.

A subset ρ_i of rows of a matrix "b" is called a covering subset if for every column j of matrix π there is (some) row in ρ_i which contains a "1" in column j. A covering set ρ is called unexcessive if each of its own subset is not a covering one. From now on we are interested only in unexcessive covering sets.

Lemma 1.12. Let ρ_i be an unexcessive covering set for a matrix π corresponding to a basic structure $< N, D, \underset{\sim}{b}_{N,D} >$. Then if there exists a maximum zero submatrix $\underset{\sim}{b}^{*}$ such that $R(\underset{\sim}{b}^{*}) > R(\underset{\sim}{b}_{N,D})$ it follows that $\rho_i \not\subset \underset{\sim}{b}^{*}$.

Proof. Suppose that the opposite is true, i.e. $\rho_i \subseteq \underset{\sim}{b}^{*}$. It follows immediately that no rows from N belong to $\underset{\sim}{b}^{*}$ because every such row is incompatible with ρ_i. Consequently, all the rows belonging to N must be removed from B. But it leads to an old basic structure $< N, D, \underset{\sim}{b}_{N,D} >$, i.e. $\underset{\sim}{b}^{*} = \underset{\sim}{b}_{N,D}$ what is in contradiction with the initial supposition that $R(\underset{\sim}{b}^{*}) > R(\underset{\sim}{b}_{N,D})$. So it is necessary to exclude the combination ρ_i, representing the unexcessive covering for π from B. How might this be achieved? There are three different variants described below. (Note that we require of each subset of ρ_i to be a compatible one, therefore, only such covering sets are of our interest)

Variant A. ρ_i contains only a single row. In the example shown in Fig. 1.1c. we have $\rho_1 = \{3\}$ or $\rho_2 = \{4\}$. This means that if a better solution than $\underset{\sim}{b}_{N,D} = \{3,4,7\}$ exists, then it does not contain either row 3 or row 4.

In variant A it is possible to remove a row (rows) from matrix B (with corresponding columns) under the condition that, if a better solution than current one exists, then this solution will not be lost. Otherwise, it may be directly concluded that an optimal solution is found.

Variant B. ρ_i contains exactly two rows: α and β. In this case we write "1" on the intersection of row α and column β (and correspondingly on the intersection of row β and column α).

Variant C. ρ_i contains more than 2 rows. For example, suppose $\rho_i = \{ \alpha, \beta, \gamma \}$. Introduce in matrix B a new row (column) $< \alpha, \beta >$ with the following properties:

$< \alpha, \beta >$ is incompatible with row γ and α ;

$< \alpha, \beta >$ is compatible with row β ;

This ensures that $< \alpha, \beta >$ represents disjunction of rows α and β.

Additionaly we set $B(\alpha, \beta) = B(\beta, \alpha) = 1$ (i.e. make α and β incompatible). Thus we achieve the following:

(1) the possibility of the simultaneous appearance of rows α, β and γ in the solution is excluded due to the setting $\alpha \# \beta$;

(2) the possibility for combinations $\alpha\gamma$, $\beta\gamma$ and $\alpha\beta$ to be simultaneously presented in the solution is retained since the combination α, β now corresponds to $< \alpha, \beta >$, β;

(3) the possibility for combinations $< \alpha, \beta >$, β, γ and $< \alpha, \beta >$, α, γ to be presented in the solution is excluded.

The case when ρ_i contains 4 rows may be treated in a similar manner. If ρ_i contains 5 or more rows, for instance, $\rho_i = \{ a,b,c,d,e,f \}$ then we introduce new rows: <a,b>, <c,d>, <e,f>^{0}, <e,f>^{1} combining every two neighbouring rows in ρ_i into a new one in such a way that all the new rows except one <e,f> do not contain common rows. We create a number of duplicates for the last new row <e,f> equal to the number of other new rows different from <e,f>. Therefore, one new row <e,f>^{0} corresponds to <a,b> and the other <e,f>^{1} corresponds to <c,d>. For these new rows we establish the following # - relations:

$$\begin{array}{lll}
< a,b > \# < e,f >^0 & < c,d > \# < e,f >^1 & < e,f >^0 \# < a,b > \\
a \# b & c \# d & < e,f >^0 \# < e,f >^1 \\
< a,b > \# a & < c,d > \# c & < e,f >^0 \# e
\end{array}$$

$$e\#f$$
$$<e,f>^1\#e$$
$$<e,f>^1\#<c,d>$$

<i,j> - is disjunction of rows i and j;

i,j ∈ {a,b,c,d,e,f}.

Interpretation of admittable combinations of rows in this system is illustrated by the Table 1.1.

Table 1.1.

Old combination	**Interpretation**
a,b,c,d,e,f	no interpretation (it is sufficient to build 0,1-matrix coding #-relations with rows a,b,c,d,e,f, <a,b>, <c,d>, $<e,f>^0$, $<e,f>^1$)
a,b,c,d,e	<a,b>, b, <c,d>, d, e
a,b,c,d,f	<a,b>, b, <c,d>, d, f
b,c,d,e,f	b, <c,d>, d, $<e,f>^0$, f
a,c,d,e,f	a, <c,d>, d, $<e,f>^0$, f
a,b,c,e,f	<a,b>, b, c, $<e,f>^1$, f
a,b,d,e,f	<a,b>, b, d, $<e,f>^1$, f

General rule of introducing new rows: one has to combine rows from ρ in pairs and create as many copies of the last new row as there are other new rows different from the last one. Suppose one introduced new row <α, β>. Then one needs to set the following relations:

$$<\alpha,\beta>\#\alpha$$
$$\alpha\#\beta$$
$$<\alpha,\beta>\#<x,y>^1$$

where $<x,y>^i$ is a copy of the last new row <x,y> corresponding to the new row $< \alpha, \beta >$. The row $<x,y>^i$ is compatible with the other new rows except those representing other copies of <x,y>. One needs to make all copies of the last row <x,y> mutually incompatible.

Thus, we sum up as follows.

1. At every iteration we find a basic structure

$$< N, D, \underset{\sim}{b}_{N,D} >$$

2. If the algorithm must be continued then we modify matrix B by introducing new rows (columns). In order to do this we:

2a. find matrix π and its arbitrary unexcessive covering set ρ_i

2b. exclude combination ρ_i as described above.

An arbitrary unexcessive covering set of matrix π may be found by procedure with computational complexity O(mn) where m is a number of rows in π.

__The estimation of the number of iterations.__ We suppose that B is a randomly generated matrix for which we assume that:

$\tilde{n}_0$ is the mean number of zeroes in the row of matrix B;

$\tilde{n}_1$ is the mean number of ones in the row of matrix B;

$\tilde{N}^2$ is the total number of elements in matrix B;

$p = \tilde{n}_0 / \tilde{N}$ - is the probability for any pair of rows to be compatible.

The mathematical expectation of the number of zero-submatrices containing k rows (columns) (k > 0) satisfies the relation

$$\mu(k) \le C_{\tilde{N}}^{k} * p^{C_k^2} \qquad \textbf{(1. 14')}$$

Lemma 1.13. Let matrix B be given. If on the intersection of a row α (column α) and a column β (a row β) we write a "1" instead of a "0" then we obtain

$$\mu'(k) = (3/4) * \mu(k) \qquad \textbf{(1. 14")}$$

where μ'(k) - is the new mathematical expectation of the number of zero submatrices with K rows (columns).

Proof. We may divide the total set of zero submatrices with k rows (columns) into subsets $\mu_{\alpha\beta}(\mathrm{k}), \mu_{\bar{\alpha}\beta}(\mathrm{k}), \mu_{\alpha\bar{\beta}}(\mathrm{k}), \mu_{\bar{\alpha}\bar{\beta}}(\mathrm{k})$, where the indices αβ identify those matrices from the whole number μ (k) which contain rows α and β. Inversion $\bar{\alpha}$ ($\bar{\beta}$) means that row α (β) does not belong to the given subset of matrices.

In a probabalistic sense, rows α and β are indistinguishable, i.e.

$$\mu_{\alpha\beta}(k) = \mu_{\bar{\alpha}\beta}(k) = \mu_{\alpha\bar{\beta}}(k) = \mu_{\bar{\alpha}\bar{\beta}}(k).$$

Corollary. If an element B(i,j) = 0 then after setting B(i,j) = 1 every mathematical expectation μ (2), μ (3),..., μ (k) will reduce according to equation (1.14").

We are interested in the maximum k such that μ (k) ≥ 1/2 and μ (k+1) < 1/2. This maximum k determines a probabalistic estimation of the number of maximum zero submatrices of matrix B. To be more accurate, we denote such a maximum k as $\mathrm{k}^{\max}$. We will use a rule of "3 σ" which defines that any random value x belongs to the diapason μ ± 3σ with a probability rather close to 1, where μ is mathematical expectation of x and σ - standard deviation,

$$\sigma = \sqrt{\mu\left(k^{\max}\right)\left(1 - p^{C_k^2}\right)} \approx \sqrt{\mu\left(k^{\max}\right)}$$

Further we suppose

$$\tilde{\mu}\left(k^{\max}\right) = \mu\left(k^{\max}\right) + 3 \cdot \sqrt{\mu\left(k^{\max}\right)\left(1 - p^{C^2_{k^{\max}}}\right)} \qquad \textbf{(1.14''')}$$

where μ ($\mathrm{k}^{\max}$) is determined from (1.14').

Lemma 1.14. If we suppose that after every iteration the new value of $\bar{\mu}$ ($\mathrm{k}^{\max}$) becomes equal to 3/4 $\tilde{\mu}$ ($\mathrm{k}^{\max}$), where $\tilde{\mu}$ (.) is the estimation of the number of zero submatrices with k^{max} rows before iteration, we can draw a conclusion that, in order to reduce $\tilde{\mu}$ (k^{max}) to 1/2, one needs to repeat Z iterations, where

$$Z = \frac{\ln\left(2\tilde{\mu}\left(k^{\max}\right)\right)}{\ln 4 - \ln 3}. \tag{1. 15}$$

Proof. Indeed, from

$$\tilde{\mu}\left(k^{\max}\right)\cdot\left(1/4+3/16+9/64+27/256+...+3^{z-1}/4^{z}\right)=\tilde{\mu}\left(k^{\max}\right)-1/2$$

one may derive

$$(3/4)^{z}=1/\left(2\tilde{\mu}\left(k^{\max}\right)\right)$$

using the formula

$$a_0+a_0q+a_0q^2+...+a_0q^n=a_0\left(1-q^{n+1}\right)/(1-q)$$

with $0 < q < 1$.

The last may be transformed to the equation in the form (1.15).

Lemma 1.15. The suggested rules for cutting an arbitrary combination of rows, given by some unexcessive covering set ρ_i, provide a contraction of $\tilde{\mu}$ (k^{max}) approximately as stated by (1.14").

Proof. Consider any new row <a,b> introduced at some iteration with new relations

$$\begin{cases} <a,b>\#a \\ \quad a\#b \end{cases} \tag{*}$$

for this row.

Obviously, by virtue of (*), the real number Z of all zero submatrices with k rows (columns) of the initial 0,1- matrix cannot change. It is the same with respect to a copy <x,y> of the last new row corresponding to <a,b>. So, an additional established relation

$$<a,b>\#<x,y>^i$$

causes contraction $\tilde{\mu}$ (k^{max}) approximately as that stated by (1.14").

Note, however, that all these considerations are valid with regard to a randomly generated 0,1-matrix containing the same number of units (zeroes) and having the same dimensions as the considered one. So the above estimations remain correct in a probabalistic sense. The last point to be clarified is the dependancy of $\tilde{\mu}$ (k^{max}) on a new row < a,b > added to the matrix. Remembering, that <a,b> is a disjunction of the rows a and b with an additional "1" due to the relation <a,b> # a, we now show that adding this type of new row to a given matrix for arbitrary a and b will cause a decrease in the value of $\tilde{\mu}$ (k^{max}).

Let N_0 be the number of all zeroes in the 0,1-matrix B and μ represent the number of zeroes in the disjunction of two arbitrary rows <a,b> from B.

We have

$$\mu < \frac{N_0^2}{N^3}.$$

If <a,b> is added to B in accordance with the established rules, then it causes the total number of zeroes to become not greater than

$$N_0 + 2\left(\frac{N_0^2}{N^3} - 1\right) - 2 \ .$$

Compare the values $\mu_1 = C_N^{k^{max}} \cdot P_0^{C_{k^{max}}^2}$ and $\mu_2 = C_{N+1}^{k^{max}} \cdot P_0^{C_{k^{max}}^2}$, where $P_0 = \frac{N_0}{N^2}$ and $P_1 = \frac{N_0 + 2\left(\frac{N_0^2}{N^3} - 1\right) - 2}{(N+1)^2}$. Namely, let us show that $\mu_1 \geq \mu_2$.

This means that from the probabalistic viewpoint, adding an additional disjunctive row <a,b> to B, will cause a decreasing mathematical expectation μ (k^{max}).

After some transformations we find that the following relationship is to be proved:

$$\frac{N+1-k^{\max}}{N+1} \geq \left[\frac{\left(N^2 + \frac{2N_0}{N} - \frac{4N^2}{N_0}\right)}{(N+1)^2}\right]^{C^2_{k^{\max}}}$$

As it is rather difficult to derive an analytical proof of the above statement, we give the table below with the necessary results calculated over the reasonable diapasons of the parameters N,N_0 and k^{max}. The sign "+" denotes the case where the above relationship holds. Note also that negative results (denoted by "-") are practically meaningless due to the very low probability of the cases they represent.

N	20								
N_0	20%			60%			80%		
k	0,2N	0,5N	0,7N	0,2N	0,5N	0,7N	0,2N	0,5N	0,7N
"+"	+	+	+	+	+	+	–	+	+

N	50								
N_0	20%			60%			80%		
k	0,2N	0,5N	0,7N	0,2N	0,5N	0,7N	0,2N	0,5N	0,7N
"+"	+	+	+	+	+	+	+	+	+

N	200								
N_0	20%			60%			80%		
k	0,2N	0,5N	0,7N	0.2N	0,5N	0,7N	0,2N	0,5N	0,7N
"+"	+	+	+	+	+	+	+	+	+

N	10				15			
N_0	25%		70%		25%		70%	
k	0,2N	0,7N	0,2N	0,7N	0,2N	0,7N	0,2N	0,7N
"+"	+	+	+	+	+	+	–	+

N	25				30			
N_0	25%		70%		25%		70%	
k	0,2N	0,7N	0,2N	0,7N	0,2N	0,7N	0,2N	0,7N
"+"	+	+	+	+	+	+	+	+

N_0 - percentage of zero-elements in matrix

"+" - denotes the cases with the realized relationship

Consider the example in Fig. 1.3a. Here $\tilde{N} = 13$, $\tilde{n}_0 = 9.46$, p = 0.72769. After first iteration we obtain:

$$\left\langle N = \{2,3,5\};\quad D = \{13,10,1,9\};\quad \underset{\sim N,D}{b} = \{4,6,7,8,11,12\} \right\rangle$$

$$\mu(k+1) = \mu(7) = C_{13}^{7} \cdot p^{C_7^2} = 2{,}2;\quad \tilde{\mu}(7) = 7$$

$$Z = \frac{\ln 14}{\ln 4 - \ln 3} \approx 9{,}2\ .$$

	1	2	3	4	5	6	7	8	9	10	11	12	13
1		1		1			1			1			
2	1		1			1				1	1		
3		1					1	1					1
4	1								1				
5						1				1	1	1	
6		1			1								1
7	1		1							1			
8			1						1				1
9				1				1			1		
10	1	1			1		1					1	
11		1			1				1				
12					1					1			1
13			1			1		1				1	

Fig. 1. 3. a)

	2	3	5
1	1	0	0
2	0	1	0
3	1	0	0
4	0	0	0
5	0	0	0
6	1	0	1
7	0	1	0
8	0	1	0
9	0	0	0
10	1	0	1
11	1	0	1
12	0	0	1
13	0	1	0

Fig. 1. 3. b)

To modify matrix B we build matrix π (Fig.1.3b) and find

$\rho_1 = \{6,7\}$; $\rho_2 = \{11,8\}$; $\rho_3 = \{11,7\}$; $\rho_4 = \{6,8\}$.

Now we set up B(6,7) = B(7,6) = B(11,8) = B(8,11) = B(11,7) = B(7,11) = B(6,8) = =B(8,6) = "1". The second iteration results in (Fig.1.3c):

	1	2	3	4	5	6	7	8	9	10	11	12	13
1		1		1			1			1			
2	1		1			1				1	1		
3		1					1	1					1
4	1								1				
5						1				1	1	1	
6		1			1		1	1					1
7	1		1			1				1	1		
8			1			1			1		1		1
9				1				1			1		
10	1	1			1		1					1	
11		1			1		1	1	1				
12					1					1			1
13			1			1		1				1	

Fig 1. 3. c)

$\tilde{N}=13$, $\tilde{n}_0=8.84$, p=0.68, $\tilde{\mu}(7)=2.7$, Z=6.3 .

There is an unexcessive covering set $\rho_1 = \{3\}$, so it is necessary to delete row 3 and column 3. Therefore we obtain

$$\left\langle N = \{6,10,11\};\quad D = \{1,9,13,5\};\quad b_{\sim N,D} = \{2,4,7,8,12\} \right\rangle.$$

Having built matrix π and removed row 2 and column 2, we obtain $\tilde{\mu} < 1$; i.e. procedure stops with the best record $\underset{\sim}{b}^* = \{4,6,7,8, 11,12\}$.

Remark. We must remember that this result is optimal in a "probabilistic sense".

Conclusions. Obviously the suggested algorithm is rather effective since $Z=O(\ln\mu)$, where μ is the mathematical expectation of the maximum number of zero submatrices. In comparison with Geoffrion's algorithm and Balas' algorithm it proves to be significantly faster for the number of individual tasks requiring computational time equal to (0,006-0,3) t_{comp}, where t_{comp} - is an analogous time required by the above mentioned algorithms.

Note that matrix size growth function has also the form

$O(|R|\ln\mu)$ where $|R|$ represents the number of rows in $\underset{\sim}{b}^*$. Indeed, the number of new rows added to B at every iteration is not bigger than (R-2) and the number of iterations is about $O(\ln\mu)$.

1.5. On the minimum-size cover problem (MSCP)

Let $B_{n,m}$ be a 0,1-matrix with n rows and m columns. Suppose that $B_{n,m}$ has no all zero column.

Let us say that row i covers column j in $B_{n,m}$ if i contains a "1" in column j. A set Π of rows is said to be an unexcessive covering set of the matrix $B_{n,m}$, if each column from $B_{n,m}$ is covered by at minimum one row from Π and any subset of Π does not satisfy this condition.

The minimum size cover problem (MSCP) consists in finding a set B^* of rows characterized by the following:

(i) it contains the minimum number of rows among all the possible covering sets;

(ii) for every column j in $B_{n,m}$ there is a row α from B^* such that B(α,j)=1 (i.e. the row α covers column j).

A covering set is an unexcessive one if it is not possible to delete any row without violating point (ii) above.

The following lemmas are obvious and need no proofs.

Lemma 1.16. Let α and β be two rows such that in every column, where β contains "1", α contains "1" also. Then row β may be deleted without losing a solution.

Lemma 1.17. Let α and β be two columns such that in any row, where α contains "1", β contains "1" also. Then column β may be deleted from $B_{n,m}$ without loss of solution of the initial MSC-problem.

Lemma 1.18. If there exists a column j containing the only "1" in the row α then this row α should be included in B^*.

Lemma 1.19. If row α is known to be included in B^* then one may delete all the columns, containing "1" in the row and retain at minimum one solution in the resulting 0,1-matrix.

Let Π_i be an arbitrary unexcessive covering set for $B_{n,m}$, Π_i ={α,β,...,γ}. Then there exists a column j1 in which row α contains a "1" and rows β,...,γ contain only zeroes; there exists also a column j2 in which row β contains a "1" and rows α,...,γ contain only zeroes, etc.

Definition. Let Π_i be an arbitrary unexcessive covering set for $B_{n,m}$, Π_i = {α,β,..., γ} and let B(α,j) = 1,B(β,j)= 0 = ... =B(γ,j)= 0 for some column j. The element B(α,j) on the cross of row α and column j will be called a symptomatic (characteristic) element of Π_i or simply a symptomatic (characteristic) element.

Lemma 1.20. Every unexcessive covering set has a unique set of all its symptomatic elements (the last set will be called the *syndrome*).

Proof. Let there be given two different unexcessive covering sets Π_1 and Π_2 and two rows α and β such that $\alpha \in \Pi_1$, $\alpha \notin \Pi_2$, $\beta \in \Pi_2$, $\beta \notin \Pi_1$. It follows from the definition above that there exists a column k with symptomatic element B(α,k) and a column l with symptomatic element B(β,l), i.e., corresponding syndromes of Π_1 and Π_2 are different due at least to these symptomatic elements.

An algorithm for solving MSC-problem is based on the following scheme;

(S1). Realizing rather a good heuristic algorithm to obtain some unexcessive covering set.

(S2). Making column-resolvent(s) (consequence(s)) from the supposition that the covering set obtained is not minimal. It will be clear that on the basis of generated resolvents one would be able to get the minimum-size covering set or repeat step S1 with increasing probability of finding a required cover.

(S3). Repeating the above given steps (if a minimum-size covering set is not obtained) with the modified matrix $B_{n,m'}$. $B_{n,m'}$ is obtained from $B_{n,m}$ by including these column-resolvents).

To find some unexcessive covering set a kind of a "greedy" algorithm may be used which prefers to include in the covering set those rows which contain the maximum number of "1"s in the considered 0,1-matrix.(If a row is included in the cover then all the columns, covered by α, should not be taken into account when selecting other rows to the covering set).

Consider now p. S2. Assume for the matrix in Fig 1.4a, the first unexcessive covering set $\Pi = \{\beta_3, \beta_4, \beta_5, \beta_6\}$ has been obtained. Using the given set Π we can produce a set of new columns which are to be added to $B_{n,m}$ in accordance with the following theorem.

Theorem 1.3. Let the unexcessive covering set Π of matrix $B_{n,m}$ contain k rows. Choose k arbitrary and different columns $j_1,...,j_k$. For these k columns find a set A of row numbers each of which contains more than a single "1" in columns $j_1,...,j_k$.

i) if A = $\varnothing$, then Π is an optimal minimum-size covering set;

ii) if A$\neq \varnothing$, then if Π is not a minimum-size covering set, at least one of the rows from A necessarily belongs to each minimum-size covering set.

	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}	α_{11}	α_{12}	α_{13}
β_1	1	1					1		1				
β_2		1				1							
β_3	1		1				1	1					1
β_4									1		1	1	
β_5		1	1			1				1			
β_6				1	1			1					
β_7		1					1				1		1
β_8				1		1				1			
β_9					1		1				1	1	
β_{10}	1	1			1								
β_{11}				1		1		1		1			

Fig. 1. 4 a)

Thus, for Fig.1.4a choose, for example, columns α_1, α_2, α_3, α_4 and find A = {β_1, β_3, β_5, β_{10}}. The theorem asserts that if Π ={β_3, β_4, β_5, β_6} is not a minimum-size covering set then every minimum-size covering set contains at minimum one row from A.

Proof may be obtained without difficulties.

Corrolary 2. If A≠ ∅ then one can add to $B_{n,m}$ a new column containing "1" in the intersection with every row belonging to A.

Corrolary 3. If Π is not optimal then there is a set A which has no common rows with Π.

Proof. If such a set A does not exist, then it is correct, that Π is an optimal solution by virtue of theorem 1.2, in respect to some nonminimal covering set with the same cardinal number. It comes to a contradiction.

Let us again consider Fig.1.4a. Generate the cuttings A_1, A_2, A_3, A_4, A_5 which correspond to definite combinations of columns of the initial matrix $B_{n,m}$, namely A_1 = {β_3, β_4 } corresponds to columns α_8, α_9, α_{12}, α_{13}; A_2 = {β_3, β_{11}} corresponds to columns α_6, α_8, α_{12}, α_{13}; A_3 = {β_3, β_9 } corresponds to columns α_3, α_5, α_{12}, α_{13}; A_4 = {β_3 }corresponds to columns α_3, α_8, α_{12}, α_{13}; A_5 = {β_5, β_9 } - α_3, α_5, α_{10}, α_{12} (Fig.1.4b).

We call the following scheme

	A_1	A_2	A_3	A_4	A_5
β_1					
β_2					
β_3	1	1	1	1	
β_4	1				
β_5					1
β_6					
β_7					
β_8					
β_9			1		1
β_{10}					
β_{11}		1			

Fig. 1. 4 b)

$$\alpha_{i1}, \alpha_{i2}, \dots , \alpha_{ik} \vdash A_i$$

resolution scheme bearing in mind that the set A_i generated by virtue of given columns α_{i1}, α_{i2} , ... , α_{ik} represents their consequence. The strategy of generating the new columns A_i will be further called a resolvent generating discipline (RGD). The designation RGD(A)=B means that B is a set of resolvents produced from A.

From Fig 1.4b it is clear that row β_3 must be included into the optimal solution. So we can apply lemma 1.19 to the initial matrix in Fig.1.4a (see result in Fig.1.5). For the latter we may produce the following resolvents for the columns in Fig.1.5 (now in RGD only three columns take place as one row β_3 is already included in solution found), $A_7 = \{\beta_6\}$ for the columns $\{ \alpha_4, \alpha_5, \alpha_9 \}$, $A_8 = \{\beta_4, \beta_9 \}$ for the columns $\{ \alpha_9, \alpha_{10}, \alpha_{12} \}$. The row β_6 should also be included into the optimal solution, etc. The final result $\Pi =\{\beta_3, \beta_6, \beta_4, \beta_5\}$ is really a minimum-size covering set.

The RGD, in respect to the current matrix $B_{n,m}$ and the best covering set found, results in one of the following issues:

Issue A. It is proved that Π is an optimal covering set.

Issue B. A better covering set than Π is obtained.

Issue C. Some new additional columns are generated which do not belong to $B_{n,m}$.

In the case "Issue B" one needs to repeat p.S2. The case "Issue C" should be further examined.

	α_2	α_4	α_5	α_6	α_9	α_{10}	α_{11}	α_{12}
β_1	1				1			
β_2	1			1				
β_3								
β_4					1		1	1
β_5	1			1		1		
β_6		1	1					
β_7	1						1	
β_8		1		1		1		
β_9			1			1	1	1
β_{10}	1		1					
β_{11}		1		1				

Fig. 1.5.

First of all let us show that, having some unexcessive set Π with k rows, we can always generate additional columns which do not belong to B_{n,n_1} and cannot be excluded due to deterministic lemmas 1.16-1.19.

Define a set of all symptomatic elements of the covering set and the columns which contain them.We call these columns characteristic. It is obvious (due to lemma 1.20) that the number of such columns cannot be less than k. Let, for definitness, they will be $\alpha_{i1}, \alpha_{i2}, \dots, \alpha_{ik}$ and there are no two symptomatic elements in the same row. Let us find resolvent A from

$$\alpha_{i1}, \alpha_{i2},\dots, \alpha_{ik} \vdash A.$$

Then A is the very column we are speaking of, i.e. A cannot be excluded by lemma 1.17. Really, A has no row common with Π. If we assume that A should be excluded by lemma 1.17, then there must be some column β which also has no common units in the rows from Π. But in this case Π cannot cover column β. This comes to a contradiction. We may derive from this consideration the following.

Assertion. For a given unexcessive covering set Π with k rows one can introduce at least one new column-resolvent, which, by the way, cuts the given covering set Π.

Adding a new column resolvent:

- does not change the number of minimum-size covering sets in matrix ;
- reduces the number of non-minimal covers of the initial matrix;

- reduces the value of the mathematical expectation of the number of minimum-size covering sets ;

For the issue "C" above we can use the following assertion: an optimal solution of the given **MSCP** is provided by the finite number of the resolvent-columns added. Evidently, we are highly interested in the procedure which generates "good" resolvents. We conclude this paragraph with a depiction of such a procedure. It finds for the given k rows from unexcessive covering set $\Pi = \{\alpha_{i1,}\alpha_{i2,...,}\alpha_{ik}\}$ columns $\{\beta_{j1},\beta_{j2},...,\beta_{jk}\}$ producing the desired resolvent. The procedure we are interested in consists of the following rules:

1. Consider current matrix B. If there remain only k columns then find their resolvent. Stop.

Otherwise perform the next step.

2. If there is a row β containing one or less "1"s - delete β. Go to step 3.

3. Find the column with maximum number of "1"s and delete it. Repeat from step 1.

This common scheme may have different particular implementations. For example, for the matrix in Fig.1.4a and k=4 it may produce the following resolvent $A = \{\beta_3\}$ by deleting the columns and the rows in the following order: $\{\alpha_2, \beta_2, \alpha_7, \alpha_9, \beta_1, \alpha_4, \alpha_5, \beta_6, \beta_{10}, \alpha_6, \beta_8, \alpha_{10}, \beta_{11}, \beta_5, \alpha_{11}, \beta_4, \beta_7, \beta_9, \alpha_1\}$.

Thus, our solving strategy for the **MSCP** is essentially based on a cutting mechanism with rather a high practical efficiency.

1.6. Precedence and incompatibility

Let the set of e.s.o be finite with a precedence relation ($\succ$) given on it. The precedence relation may be interpreted by a graph G with vertices $\{O_i\}$ and arcs $(\overrightarrow{O_i, O_j})$ such that $O_i \succ O_j$. Also we suppose that there is #-relation defined on $\{O_i\}$.

Let us address the next example:

$$O_1 \succ O_2, O_1 \succ O_3, O_4 \succ O_2, O_4 \succ O_5, O_2 \succ O_6, O_3 \succ O_7,$$

$$O_3 \succ O_8, O_1 \succ O_7, O_4 \succ O_6, O_5 \succ O_7, O_4 \succ O_7, O_4 \succ O_3;$$

It is required to find a maximum-size sequence K of e.s.o., satisfying the given relations, i.e. if O_i occupies a more left position in K than O_j it should be $O_i \succ O_j$ and, besides, no two e.s.o in the sequence K are incompatible. It is also presumed that every two e.s.o O_i and O_j not connected by some route in graph G are incompatible.

The idea is in replacing precedence-relations by the #-relations. As a result we will obtain a maximum-size zero sub- matrix problem. To realize the idea create the next table

Table 1. 2.

Vertex (1) (e.s.o)	The formed sets of linked e.s.os (2)
O_1	$+O_2$, $+O_3$, $+O_6$, $+O_7$, $+O_8$
O_2	$-O_1$, $-O_4$, $+O_6$
O_3	$-O_1$, $+O_7$, $+O_8$, $-O_4$
O_4	$+O_2$, $+O_5$, $+O_3$, $+O_7$ O_8
O_5	$-O_4$, $+O_7$
O_6	$-O_1$, $-O_4$, $-O_2$
O_7	$-O_1$, $-O_3$, $-O_5$, $-O_4$
O_8	$-O_1$, $-O_3$

The procedure for making the above table is the following. Consider all the $\succ$ - relations starting with $O_1 \succ O_2$ Write in O_i - row of the table 1.2 term "+ O_2 " and in O_2 - row - term "- O_1 " bearing in mind that "+ O_i " in the row O_j means $O_j \succ O_i$ and "- O_i " in the row O_j means $O_i \succ O_j$. For the relations $O_1 \succ O_3$, $O_4 \succ O_2$, $O_4 \succ O_5$ proceed in the same way. When considering relation $O_2 \succ O_6$, write in the row O_6 term "- O_2 " and all negative terms in the row O_2. Write in the row O_2 term "+ O_6 ". It is clear that in this way we shall find the transitive closure of the origin graph G [13,15], so all remaining actions are omitted. The general rule for filling up the table may be easily deduced.

Now we immediately obtain an 0,1-matrix by coding the #-relations (Fig.1.6) with elements b_{ij} defined as follows:

b_{ij} = 0, if in the row O_i of the table 1.2 there is a member - O_j or in the row O_j there is a member - O_i and $O_i \overline{\#}$ O_j.

	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8
O_1					1	1	1	
O_2			1	1	1		1	1
O_3		1		1	1	1		
O_4		1	1			1		1
O_5	1	1	1			1	1	1
O_6			1	1	1		1	1
O_7	1	1			1	1		1
O_8		1		1	1	1	1	

Fig. 1. 6.

Thus, if b_{ij} = 0 then $O_i \overline{\#}$ O_j (i.e., O_i and O_j are compatible).

b_{ij} = 1 in other cases including those given by the initial #-relations, i.e.

$O_1 \# O_7, \quad O_4 \# O_6, \quad O_4 \# O_7, \quad O_4 \# O_3.$

Solving maximum size zero submatrix problem on matrix shown in Fig. 1.6. we obtain a maximum-size zero-submatrix $\langle O_1, O_3, O_8 \rangle$.

Assertion. The above procedure finds (one of the) maximum-size route consisting of only mutually compatible vertices.

Proof. In fact, it is only required to prove that the resulting set of vertices form some route in the graph G. Suppose, that the opposite is true. Then, there are two vertices O_i and O_j such that $\neg(O_i \succ O_j)$ and $\neg(O_j \succ O_i)$. But it leads to b_{ij} = 1, and, therefore, O_i # O_j. So, we obtain a contradiction.

1.7. Prohibition

We shall say that e.s.o. O_i prohibits e.s.o. O_j and write this fact as $O_i \multimap O_j$ if O_j cannot be executed after O_i. That is, if $O_i \multimap O_j$ then the solving sequence K = < ..., O_i ,..., O_j ,...> is invalid, although it is not always rightful in relation to the sequence K' = <..., O_j ,..., O_i ,...>. One may see that a prohibition is not "pure" logical relation since it depends on the mutual arrangment of two elementary solving operators.

For example, let the following system be given:

a —o b c —o e c —o b
a —o e f —o b a —o c
f —o a e —o d.

There are three main problems one may face, i.e.

(Ai) to find a valid sequence K with all e.s.o. presented in it;

(Aii) to find a maximum-length valid sequence K;

(Aiii) to find a minimum-length valid sequence K which cannot be extended without loss of validity.

Let us consider the first two problems. Introduce graph G(U, V) with the vertices {a,b,c, d,e,f} and the arcs $\vec{V} \ni \{ \overrightarrow{x,y} \}$ provided x —o y ; x,y ∈ U. For our example, G(U,$\vec{V}$) has the form shown in Fig. 1.7a. Begin with the problem (Ai).

It is clear that every vertex not containing emanating arcs may be assigned to the leftmost position in K. In the considered example there are two such vertices "b" and "d" which are included first in K in the arbitrary order:

$$K = \langle b, d, \ldots \rangle.$$

Modify graph G(U,$\vec{V}$) by deletion of both vertices "b" and "d" with the coherent arcs (Fig. 1.7b). Thus, we obtain a new candidate, namely vertex "e":

$$K = \langle b, d, e \ldots \rangle.$$

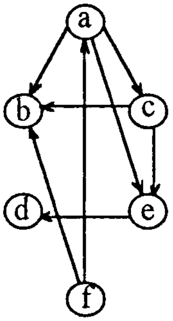


Fig. 1. 7 a)

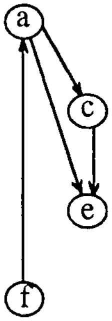


Fig. 1, 7 b)

proceeding by analogy we obtain

$$K = \langle b,d,e,c,a,f\rangle.$$

The whole idea is rather clear. However, it is, evident that problem (Ai) is unsolvable when the graph G(U,$\vec{V}$) contains cycles since the cycle excludes some e.s.o. from the resulting sequence K.

Problem (Aii) comes to the following one: Find the minimum number of vertices which, if deleted, provide no cycles in the graph G(U,$\vec{V}$). The problem (Aii), therefore, is a kind of minimum-size cover problem considered earlier. Thus, for the system

a —o b, b —o c, c —o a, b —o d, d —o e, e —o f, f —o d

we have a graph G(U,$\vec{V}$) shown in Fig. 1.8 and a minimum-size covering set of vertices {c,e} involved in two different cycles. Deleting both vertex "c" and vertex "e" we obtain an acyclic graph G which directly suits (Ai)-problem. The (Aii)-problem may be formalized in terms of solving #-equations. Consider, for example, the relation

a —o b.

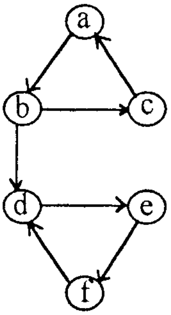


Fig. 1. 8.

To represent this in terms of the incompatibility relation let us introduce a new object -r_a standing for the "right side of "a" in sequence K". Thus, we obtain directly

r_a # b

instead of the above relation a —o b. The initial system of equations may be replaced by

$$r_a \# b \quad r_c \# a \quad r_d \# e \quad r_f \# d$$

$$r_b \# c \quad r_b \# d \quad r_e \# d$$

$$r_a + r_b + r_c = 1 \qquad\qquad r_e + r_f + r_d = 1$$

with criterion

a + b + c + d + e + f → max.

1.8. Conditional executability

Let there be given a logical equation in the conjunctive form as that shown below

$$L_1 = \left(a\,\bar{b} \vee \bar{c}\,e\right)\left(b\,\bar{c} \vee c\,\bar{a}\right)\left(a\,d \vee \bar{b}\,e \vee \bar{c}\right)\left(\bar{e} \vee \bar{c}\right) = 1\,.$$

We are interested in finding a valid solution of this equation with the maximum number of positive boolean variables (i.e. variables without negation). There are some extreme cases of this problem. The first one is characterized by the absense of positive symbols in the original logical equation as in the following example

$$L_2 = \left(\bar{a}\,\bar{c} \vee \bar{b}\right)\left(\bar{a}\,\bar{e}\,\bar{d} \vee \bar{b} \vee \bar{c}\,\bar{e}\right)\left(\bar{f} \vee \bar{d}\right)\left(\bar{e}\,\bar{h} \vee \bar{b}\right) = 1\,.$$

This problem is directly reduced to the minimum-size cover problem with equivalent formulation on the sets represented below

$$A = \begin{Bmatrix} \bar{a}\,\bar{c} \\ \bar{b} \end{Bmatrix},\quad B = \begin{Bmatrix} \bar{a}\,\bar{e}\,\bar{d} \\ \bar{b} \\ \bar{c}\,\bar{e} \end{Bmatrix},\ C = \begin{Bmatrix} \bar{f} \\ \bar{d} \end{Bmatrix},\ D = \begin{Bmatrix} \bar{e}\,\bar{h} \\ \bar{b} \end{Bmatrix}$$

and the optimum solution

$$S^{opt}=\left\{a,c,e,f,h,\overline{d,b}\right\}$$

with minimum-size covering set $S^{c}=\left\{\overline{d},\overline{b}\right\}$.

In the other extreme case there is at least one elementary conjunction having no negative symbols in every disjunctive form participating in the representation of L, i.e.

$$L=f_1\ \&\ f_2\ \&\ldots\&\ f_n\,.$$

Thus, in the example below

$$L_3=(\overline{a}b\vee c)(a\vee\overline{d}\overline{e})(\overline{b}c\vee\overline{c}e\vee dc)=1$$

$f_1=\overline{a}\,b\vee c$ with positive conjunction "c"

$f_2=a\vee\overline{d}\,\overline{e}$ with positive conjuction "a"

$f_3=\overline{b}\,c\vee\overline{c}\,e\vee d\,c$ with positive conjuction "$d\,c$".

In this case the solution contains all the symbols (without negation) used in L, i.e.

$$S^{opt}=\left\{a,b,c,d,e\right\}.$$

It is possible sometimes to transform an original equation to one of the above cases. For example, if

$$L_4=\left(a\,\overline{b}\vee\overline{c}\,e\right)\left(b\,\overline{c}\vee c\,\overline{a}\right)=1$$

one may extend L_4 to the next equivalent form

$$L_4=\left(a\,\overline{b}\vee\overline{c}\,e\vee\overline{b}\,\overline{c}\right)\left(b\,\overline{c}\vee c\,\overline{a}\right)=$$

$$=\left(a\,\overline{b}\vee\overline{c}\,e\vee\overline{b}\,\overline{c}\right)\left(b\,\overline{c}\vee\overline{a}\right)=$$

$$=\left(\overline{c}\,e\vee\overline{b}\,\overline{c}\right)\left(b\,\overline{c}\vee\overline{a}\right)=\overline{c}\left(e\vee\overline{b}\right)\left(b\vee\overline{a}\right)=1$$

and deduce from it the solution

$$S^{opt}=\left\{a,b,e,\bar{c}\right\}.$$

Let us return to the formula L$_1$, and denote $A=\left\{a,b,c,d,e\right\}$, where A is the set of all symbols contained in L$_1$. Consider the first disjunctive form $\left(a\,\bar{b}\vee\bar{c}\,e\right)$ and denote

$V_1=a\,\bar{b}\,,V_2=\bar{c}\,e.$

Thus, for disjunctive form $\left(a\,\bar{b}\vee\bar{c}\,e\right)$ we obtain the coresponding subset

$$\begin{Bmatrix}V_1=a\,\bar{b}\\ V_2=\bar{c}\,e\end{Bmatrix},$$

Acting by analogy, we produce the following subsets β_k of V_i, for every disjunctive form

$$\beta_1=\begin{Bmatrix}V_1=a\,\bar{b}\\ V_2=\bar{c}\,e\end{Bmatrix},\ \beta_2=\begin{Bmatrix}V_3=b\,\bar{c}\\ V_4=c\,\bar{a}\end{Bmatrix},\ \beta_3=\begin{Bmatrix}V_5=a\,d\\ V_6=\bar{b}\,e\\ V_7=\ \ \bar{c}\end{Bmatrix},$$

$$\beta_4=\begin{Bmatrix}V_8=\bar{e}\\ V_9=\bar{c}\end{Bmatrix}.$$

Now let us create a 0,1-matrix with the rows (columns) a, b, c, d, e, V_1,...,V_9 (Fig. 1.9) such that

1) on the intersection of row x and column y, we write "0", if both x and y are e.s.os and there is a vector V_k such that

$$\left\{\begin{matrix}x\,\&V_k\neq\\ y\,\&V_k\neq\end{matrix}\right.\square$$

where □ stands for logical falsehood

2) on the intersection of row x and column y, we write "0" if x is e.s.o. and y is a vector V_j and

$x \mathbin{\&} V_j \neq □$, i.e.,

$V_j \nleq \bar{x}$;

otherwise (i.e. if $x \mathbin{\&} V_j = □$) we write "1";

3) on the intersection of row x and column y, we write "0", if both x and y are some vectors V_i and V_j, and

$V_i \mathbin{\&} V_j \neq □$;

otherwise we write "1", i.e., if

$V_i \mathbin{\&} V_j = □$

or V_i and V_j belong to the same set β_z for some Z.

	a	b	c	d	e	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9
a									1					
b						1					1			
c							1	1				1		1
d														
e													1	
V_1		1					1	1	1					
V_2			1			1			1				1	
V_3			1			1			1		1			
V_4	1					1	1	1		1		1		1
V_5									1		1	1		
V_6		1						1		1		1	1	
V_7									1	1	1			
V_8					1		1				1			1
V_9			1						1				1	

Fig. 1. 9.

To provide a maximum-size zero submatrix of the matrix in Fig. 1.9 to be a solution b^* with the maximum number of positive symbols, it is necessary to ensure that exactly one vector V_j from every set β_k be presented for b^*. Let us designate this condition as C0. Thus $b^* = \langle a, b, d, e, V_2, V_3, V_7, V_9 \rangle$ is an optimum submatrix from which one may easily deduce the solution $\left\langle a,b,d,e,\overline{c}\right\rangle$ for the original equation L_1 = 1. To ensure condition C0 (in the case whenL_1 = 1 is solvable) one needs to assign to every vector V_i, the same weight w(V_i), which is equal to (N+1), where N is the total number of e.s.o. symbols; every symbol of e.s.o. (a, b, c, d, e) should have a weight of 1. Now if a valid solution exists, this will be obtained. This follows directly from the fact that including any V_i in b^* is equivalent to including all the e.s.os in b^*.

1.9 Other examples

Let us consider some well-known problems which may be directly solved on the basis of cutting strategies involving weak methods. Some examples are left without comments. Others contain an outline of all the necessary transformations needed to represent an initial problem in the terms of basic problems.

Example 1. The problem of a graph colouring [24,26].

Example 2. Finding a maximum-size clique in the graph [17].

Example 3. Maximum matching set problem [17,27].

Example 4. Finding a solution to a boolean equation. Let there be an equation in the form

$$\left(\overline{x_1}\,x_2 \vee x_1\,x_3 \vee \overline{x_4}\right)\left(x_1 \vee x_2\,\overline{x_3} \vee x_3\,\overline{x_4}\right)\left(x_1\,\overline{x_4} \vee x_2\right) = 1$$
$$x_i \in \{0,1\}.$$

We introduce the following sets:

$$\beta_1 = \{b_1, b_2, b_3\},\ \beta_2 = \{b_4, b_5, b_6\},\ \beta_3 = \{b_7, b_8\}$$

where

$$\begin{array}{lll} b_1 = \overline{x_1}\,x_2 & b_4 = x_1 & b_7 = x_1\,\overline{x_4} \\ b_2 = x_1\,x_3 & b_5 = x_2\,\overline{x_3} & b_8 = x_2 \\ b_3 = \overline{x_4} & b_6 = x_3\,\overline{x_4} & \end{array}$$

and β_1, β_2, β_3 stand for corresponding disjunctive forms restricted by round brackets.

Let us build a 0,1-matrix A such that $a_{ij} = 0$ if b_i and b_j satisfy the following conditions

- b_i and b_j do not contain common elements

or

- b_i & b_j $\neq$□ (their conjunction is not "false").

The last condition means that there is no object x_k such that $x_k \in b_i$ and $\overline{x_k} \in b_j$ or vice versa.

In addition, we set $a_{ij} = 1$ for b_i and b_j such that there is an index q for which $b_i \in \beta_q$ and $b_j \in \beta_q$.

As a result the matrix shown in Fig.1.10 would be obtained. A maximum zero submatrix of this 0,1-matrix is $b^* = \{b_3, b_5, b_8\}$. It is easy to see that b^* determines a suitable solution for a prime boolean equation by setting $x_3 = 1$, $x_4 = 0$, $x_2 = 1$ with an arbitrary value of x_1.

	b1	b2	b3	b4	b5	b6	b7	b8
b1		1	1	1			1	
b2	1		1		1			
b3	1	1						
b4	1				1	1		
b5		1		1		1		
b6				1	1			
b7	1							1
b8							1	

Fig. 1. 10.

It is easy to conceive that if the number of rows in b^* is less then the number of different disjunctive forms then the initial boolean equation is unresolvable.

Example 5. Finding a minimum-size set of rows and columns covering all the units of a given 0,1-matrix.

According to the well known Konig's theorem this problem is equivalent to finding the maximum number of "1" in 0,1-matrix in which no pair of "1" belong to the same row or to the same column.

Example 6. An optimum assignment problem [17].

Example 7. Finding maximum-length route (cycle) in a given graph connecting (all) its vertices without repetition.

Let there be an oriented graph $G(U, \vec{D})$ (Fig.1.11a). The arcs of $G(U, \vec{D})$ are marked by the labels d_i. To proceed we must build a 0,1-matrix A with the elements $a_{ij} = 1$ if and only if the arcs d_i and d_j are the input arcs for the same vertex, or d_i and d_j are the output arcs for the same vertex, or d_i and d_j form a simple cycle on two vertices (as for arcs d_8 and d_9 in this example). Otherwise $a_{ij} = 0$. A maximum zero submatrix for our example contains the rows d_1, d_2, d_5, d_7 and corresponds to the graph presented in Fig.1.11b.

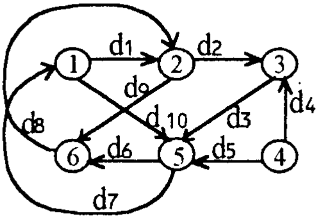


Fig. 1. 11. a)

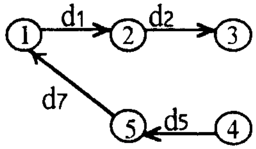


Fig. 1. 11. b)

Example 8. Boolean function minimization. This problem may be formulated as finding a minimum size covering set for a given set of simple implicants.

Example 9. Minimizing pseudo-boolean equations [22].

1.10. Conclusion

We have demonstrated an efficient technique to solve a wide-spread domain of NP-problems. We have also formulated a number of properties relevant to discrete optimization problems and then considered the basic three of them. Evidently, it is not possible to consider all possible problems here. So we are mainly interested in common principles which are useful from the problem solving system viewpoint.

Chapter 2

SOLVING DISCRETE OPTIMIZATION PROBLEMS ON THE BASIS OF Ψ-TRANSFORM METHOD

Abstract

The very nature of discrete optimization problems presumes the utilization of weak methods. Actually, such methods as dynamic programming and branches-and-bounds procedures are well-known examples of this kind. In this chapter, we shall demonstrate another approach based on a weak method strategy. This approach is a discrete modification of the Ψ-transform technique, the efficiency of which is approved by the results of solving a broad class of applied problems.

The general formulation of the problems we shall deal with is as follows

$$F(\mathbf{x}) \rightarrow \min \tag{2.1}$$

$$\mathbf{x} \in D \tag{2.2}$$

where F: $R^m \rightarrow R$, $F(\mathbf{x}) > 0$; D is a finite set of points $\mathbf{x} \in R$.

Since the type of function F(**x**) is not identified, the problems outlined in terms of (**2.1**, **2.2**) will be called F-indefinite problems. We will also identify a type of D-indefinite problem if the characterization function Θ (D) for set D is unknown.

The general idea of the Ψ-transform technique is in finding a good approximation to the characterization function Θ(D) in order to obtain the estimation of a normalized weighted value Ψ_D of the set D cardinality.

For the sequence of decreasing values $\varsigma_1 \geq \varsigma_2 \geq \ldots \geq \varsigma_n$ one needs to obtain the values Ψ_{Di} ($i = \overline{1,n}$) of the set Di cardinality ,where Di = $\{ \mathbf{x} \mid F(\mathbf{x}) \leq \varsigma_i \}$. The values of Ψ_{Di} may be estimated statistically or may be approximated analytically. It is clear that if ς^* provides a minimum of F(**x**) then $\Psi(\varsigma^*)$ becomes zero. So, all theoretical considerations are concentrated around the technique of the representation and formalization of the functions Θ(.) and Ψ(.).

2.1. Ψ-transform method

The discrete Ψ-transform method consists of substituting a unimodal continuous function $\Psi(\zeta)$ of one variable ζ, $\zeta > 0$ for a multimeasured and multiextreme $F(\mathbf{x})$ function. The function $\Psi(\zeta)$ is determined as a normalized weighted value of the set

$$E(\zeta) = \{\mathbf{x} \mid F(\mathbf{x}) \le \zeta\,, \mathbf{x} \in D\}$$

cardinality:

$$\Psi(\zeta) = \frac{\sum_{\mathbf{x} \in D} \rho(\mathbf{x}, \varsigma) \cdot \Theta(\mathbf{x}, \varsigma)}{|D|}$$

where

$$\Theta(\mathbf{x}, \varsigma) = \begin{cases} 1, \textit{if } \mathbf{x} \in E(\varsigma) \\ 0, \textit{otherwise} \end{cases} .$$

Theorem 2.1. Let the weighting function ρ (**x**, ζ) satisfy the conditions:

$\rho(\mathbf{x},\varsigma) \ge 0$ when $\mathbf{x} \in D$, $\varsigma > 0$;

$\rho(\mathbf{x}_*, \zeta_*) = 0$ when $\zeta_* = F(\mathbf{x}_*)$, $\mathbf{x}_* = \mathrm{Arg\,min}\, F(\mathbf{x})_{\mathbf{x} \in D}$

$\rho(\mathbf{x},\zeta_1) < \rho(\mathbf{x},\zeta_2)$ when $x \in D$, $\zeta_1 < \zeta_2$.

Then for all $\zeta \le \zeta^*$ the function $\Psi(\zeta)$ is zero , and when $\zeta > \zeta^*$, it is a continuous strictly increasing function.

From the theorem 2.1 it follows that the problem of finding a global extremum of the function F(**x**) on the set D is reduced to the determination of

$$\zeta^* = \max\,\{\zeta \mid \Psi(\zeta) = 0\,\}. \qquad \textbf{(2.3)}$$

To evaluate the co-ordinates of the point of global extremum of F(**x**) ,the components of the generalized centers of gravity $\overline{x}(\zeta) \in R^m$ of the sets $E(\zeta)$, $\zeta > 0$, i.e. weighted mean j-th components of vectors $\mathbf{x} \in F(\zeta)$, are found as follows:

$$\bar{x}_j(\varsigma) = \frac{\sum\limits_{x \in D} x_j \cdot \beta(\mathbf{x},\varsigma) \cdot \Theta(\mathbf{x},\varsigma)}{\sum\limits_{x \in D} \beta(\mathbf{x},\varsigma) \cdot \Theta(\mathbf{x},\varsigma)}$$

where the weighting function $\beta(\mathbf{x},\varsigma)$ satisfies the conditions:

$\beta(\mathbf{x},\zeta) > 0$ when $\mathbf{x} \in D$, $\zeta > 0$;

$\beta(\mathbf{x}^{(1)},\varsigma) > \beta(\mathbf{x}^{(2)},\zeta)$ when $F(\mathbf{x}^{(1)}) < F(\mathbf{x}^{(2)})$, $\mathbf{x}^{(1)}, \mathbf{x}^{(2)} \in D$.

It is clear that if $|E(\zeta^*) = 1|$ then the coordinates x_j^* of the global minimum point of F(x) are found as follows

$$x_j^* = \bar{x}_j(\varsigma^*), j = \overline{1,m}. \tag{2.4}$$

A general scheme for the discrete Ψ-transform method is implemented as an algorithm involving the following steps:

- generation of a set of random points in the feasible area D
- determination of the forecast estimates (**2.3**), (**2.4**) on the basis of the set obtained
- local optimization of the discrete variables vector of the problem (**2.1**), (**2,2**).

If the set D is a continuous one then the above formalization takes the following representation. The generalized continuous Ψ-function takes the form of (**2.5**):

$$\Psi(\varsigma) = \int\limits_D \rho(x,\varsigma) \cdot \Theta(x,\varsigma) \partial x \tag{2.5}$$

where $\rho(\mathbf{x}, \zeta)$,as earlier ,is a weighting function.

Components of the generalized centers of gravity $\mathbf{x}(\varsigma) \in R^m$ of the sets $E(\zeta)$, $\zeta > 0$ are defined as follows

$$x_j(\varsigma) = \frac{\int\limits_D x_j \cdot \rho(\mathbf{x},\varsigma) \cdot \Theta(\mathbf{x},\varsigma)\partial\mathbf{x}}{\int\limits_D \rho(\mathbf{x},\varsigma) \cdot \Theta(\mathbf{x},\varsigma)\partial\mathbf{x}}, \; j = \overline{1,n}. \qquad \textbf{(2.6)}$$

Theorem 2.2 [30]. Suppose, that F(**x**) $\in$ L_p , where L_p is the Lebesque's space and $\bar{x}(\varsigma) = \left[\bar{x}_1(\varsigma), \bar{x}_2(\varsigma), \ldots, \bar{x}_n(\varsigma)\right]$. If E($\varsigma^*$) is a coherent set , then

$$\lim_{\varsigma \to \xi} \bar{x}_j(\varsigma) = x_j^*, j = \overline{1,n}$$

where , $\mathbf{x}^* = \underset{x \in L_p}{Arg\min}$ F(**x**).

An analytical representation of $\bar{x}_j(\varsigma)$, j = $\overline{1,n}$ is given in [30] on the basis of an approximate evaluation of $\bar{x}_j(\varsigma)$, j=$\overline{1,n}$ at the distinct points and interpolation with quadratic polynomial. The introduction of the generalized values $\bar{x}_j(\varsigma)$ is connected with obtaining smoother approximating functions instead of using polynomials of high degree.

Thus, we can see that the Ψ-transform technique for solving problems **(2.1)**, **(2.2)** enables one to obtain optimizing vector $\mathbf{x}^*$ providing minimum $F(\mathbf{x}^*)$ with relative error

$$\hat{\varepsilon} = \left(\frac{F(\mathbf{x}^*) - \varsigma^*}{\varsigma^*}\right)_+ \text{ where } (Z)_+ = \begin{cases} Z, \; if \;\; Z > 0 \\ 0, \; if \;\; Z \leq 0 \end{cases}.$$

To reduce the relative error $\hat{\varepsilon}$ (if necessary) one needs to use the technique outlined in [30].

2.2. Some important cases of the analytical representation of the Ψ(ζ)-function

There are serious constraints on the analytical representation of the Ψ-function due to difficulties of calculating multimeasured intervals (**2.5**,**2.6**). In some important cases, however, it is possible to obtain the desired analytical representation.

Consider the linear programming problem of the form

$$F(\mathbf{x}) = (\mathbf{c},\mathbf{x}) \to \min \tag{2.7}$$

$$A\mathbf{x} \le \mathbf{b}, \tag{2.8}$$

where (**c**,**x**) is a scalar production of vectors **c**,**x** $\in R^n$;

A = $\left\|a_{i,j}\right\|$ is (m· n)-matrix of real coefficients; **b**$\in R^m$.

Suppose, that all variables x_j of the above problem must satisfy the following restrictions:

$$x_j \ge \alpha_j, \ \alpha_j \in R^n, j = \overline{1,n} \ \ \alpha_j \in R^n, j = \overline{1,n}$$

and the rank of the matrix **A** is equal to n.

In this case both sets S = { **x** | **Ax** ≤ **b** } and E(ζ) are convex polyhedrons and, in addition, E(ζ) is defined as follows

$$\begin{cases} Ax \le b \\ (c,x) \le \varsigma \ . \end{cases}$$

The measure Ψ(ζ) of the set E(ζ) may be calculated analytically on the basis of the iterative procedure proposed in [31]. Therefore, the problem of finding the global optimum of a function F(x) on the set S is reduced to the determination of ζ’

$$\varsigma^* = \max\{\varsigma | \Psi(\varsigma) = 0\} \tag{2.9}$$

After minimizing $\Psi(\varsigma)$ according to (**2.9**) (for example, on the basis of the Fibonacci procedure) one may find the optimum vector by solving the linear equations (**2.10**):

$$\tilde{A} \cdot x = \tilde{b} \qquad (2.10)$$

where $\tilde{\mathbf{A}}, \tilde{\mathbf{b}}$ - is a matrix of coefficients and a vector of right parts of the inequalities (**2.8**), obtained by deleting from **A** and **b** i-th components corresponding to excessive restrictions (**2.8**) for the solution ς^*. These components are found during the procedure [31] execution.

Let us give the main results concerning the definition of the measure of the set $S = \{\mathbf{x} | \mathbf{A} \cdot \mathbf{x} \le \mathbf{b}\}$, representing a convex polyhedron. Note, that $\Psi(\varsigma)$ as a measure of the set $E(\varsigma) = \{\mathbf{x} | F(\mathbf{x}) \le \varsigma, \mathbf{x} \in S\}$ is defined by analogy. We shall use the following designations:

a_i - the i-th row of a matrix **A**;

$\|\mathbf{x}\|$ - norm of the vector $\mathbf{x}, \|\mathbf{x}\| = \left(\sum_{j=1} x_j^2 \right)^{\frac{1}{2}}$

V(n,**A**,**b**) - the volume of a polyhedron S;

V_i (n-1,**A**,**b**) - the volume of a set $\{ \mathbf{x} | (a_i, \mathbf{x}) = b_i, \mathbf{A}\mathbf{x} \le \mathbf{b} \}$;

$(a_i, \mathbf{x})$ - scalar product in the space R^n.

Assertion 2.1 [31]. For every $\lambda > 0$ the following takes place:

$V(n,\mathbf{A},\lambda \cdot \mathbf{b}) = \lambda^n \cdot V(n,\mathbf{A},\mathbf{b})$.

Assertion 2.2. [31]. If V(n,**A**,**b**) $\neq 0$ then V(n,**A**,**b**) is a continuous function of the argument **b**.

Assertion 2.3. [31]. Suppose that

1) V(n,**A**,**b**) = 0;

2) V_i(n-1,**A**,**b**) = 0 for all $i = \overline{1,m}$.

Then V(n,**A**,**b**) is a differentiable function such that

$$\frac{\partial V(n,\mathbf{A},\mathbf{b})}{\partial b_i}=\frac{V_i(n-1,\mathbf{A},\mathbf{b})}{\|a_i\|}$$

The main result on the analytical representation of the convex polyhedron volume has been formulated in [31] as

Theorem 2.3. If V(n,**A**,**b**) is a differentiable function of argument b then

$$\mathrm{V}(n,\mathbf{A},\mathbf{b})=\frac{1}{n}\sum_{i=1}^{m}\frac{b_i}{\|a_i\|}\cdot \mathrm{V}_i(n-1,\mathbf{A},\mathbf{b}). \qquad \textbf{(2.11)}$$

As it is pointed out in [31], formula (**2.11**) may be applied in the case of a non-differentiable function V(n,**A**,**b**) as well. This circumstance is essentially used when performing the solving algorithm.

Assertion 2.4. [31]. If V(n,**A**,**b**) = 0, then

V_i (n-1,**A**,**b**) = 0 for all $i=\overline{1,m}$.

Assertion 2.5. [31]. Let V(n,**A**,**b**) ≠ 0, but there exist such numbers i for which V_i (n-1,**A**,**b**) = 0. Then all restrictions corresponding to the given indices i are excessive and equation (**2.11**) remains correct without including these components (b_i,a_i).

Now consider the algorithm which calculates a value V(n,**A**,**b**). Let V_i (n-1,**A**,**b**) = 0 and a_{i1} = 0. Then variable x_1 may be excluded from the system since

$x_1 = (b_i - a_{ij} \cdot x_j)/a_{i1}$.

Denote a vector not containing excessive variables by $\mathbf{x}^{(1)}$ and consider the system

$$(\tilde{a}_k,\mathbf{x}^{(1)})<\tilde{b}_k$$

which is obtained by excluding the variable x_1 and the i-th restriction. Denote the volume of a corresponding polyhedron by $\tilde{\mathrm{V}}_i(n-1,\tilde{\mathbf{A}}_1,\tilde{\mathbf{b}})$. Since

$$V_i(n-1,\mathbf{A},\mathbf{b}) = \frac{\|a_i\|}{\|a_{i1}\|}\cdot \tilde{V}_i(n-1,\tilde{\mathbf{A}},\tilde{\mathbf{b}})$$

formula (**2.11**) takes the form

$$V(n,\mathbf{A},\mathbf{b}) = \frac{1}{n}\cdot\frac{\mathbf{b}}{|a_{il(i)}|}\cdot \tilde{V}_i(n-1,\tilde{\mathbf{A}}_{l(i)},\mathbf{b}) \qquad \textbf{(2.12)}$$

where $\tilde{V}_i(n-1,\tilde{\mathbf{A}}_{l(i)},\mathbf{b})$ is a volume of the (n-1) dimensional i-th side S of polyhedron after excluding variable $x_{l(i)}$ and i-th restriction. Thus, volume V(n,**A**,**b**) may be evaluated recursively on the basis of the equation (**2.12**).

After (n-1) exclusions it yields the volume of the area corresponding to the restrictions

$$\alpha_l \cdot x^{(n-1)} \le \beta_l, \quad l = \overline{1, m-(n-1)}$$

where $x^{(n-1)}$ is a scalar value. This volume is evaluated by the formula

$$\max\{0;[\min_{\alpha_l > 0}(\frac{\beta_l}{\alpha_l}) - \max_{\alpha_l < 0}(\frac{\beta_l}{\alpha_l})]\}.$$

The algorithm for realizing this procedure is as follows.

Algorithm 2.1.

Input: **A**,**b**,N = {1,2,...,n}, M = {1,2,...,m}; S = 0.

Output: The volume VOL of polyhedron; the set M of the active restriction numbers.

```
Procedure VOL(M,N,A,b,S)
begin
if |N| = 1 then for all j $\in$ N do
```

```
VOL:= $\max\{0;[\min\limits_{a_{ij}>0}(\frac{b_i}{a_{ij}}) - \max\limits_{a_{ij}<0}(\frac{b_i}{a_{ij}})]\}$

else begin
    for all $l \in M$ do
     begin
     K := 1;
     while $a_{lk} = 0$ do K := K + 1;
       for all $i \in M, i \neq l$ do
        begin

        $\tilde{b}_i := b_i - (\frac{a_{ik}}{b_{ik}}) \cdot \frac{1}{a_{lk}}$

        for all $j \in N, j \neq k$ do $\tilde{a}_{ij} := a_{ij} - (a_{ik} \cdot a_{lj}) \cdot \frac{1}{a_{lk}}$ ;

comment : $\tilde{b}_i, \tilde{a}_{ij}$ - are the elements of vector b and matrix A;
        end
          V := VOL($\frac{M}{l}, \frac{N}{k}, \tilde{\mathbf{A}}, \tilde{\mathbf{b}}, 0$);
     if (|N| = n ) and (V = 0) then M:=M/l;
   comment: l-th restriction is excessive;
      S: = S + V;
      end
    VOL := S/|N|;
   end
end.
```

The above algorithm has several advantages, namely:

- it may be relatively easy programmed;

- it is not necessary to calculate the determinant (s) of matrix **A**;
- it is simpler than the existing algorithms;
- it enables one to find the excessiveness of the set (**2.2**) of constraints;
- when being programmed in the environment of such algebraic languages as REDUCE or FORMAC, the algorithm yields the analytical representation of $\Psi(\varsigma)$. However, in the view of the exponential growth of the number of operations necessary to calculate V(n,**A**,**b**) for linearly increasing n, practical effectiveness of the algorithm is still unknown.

Example. Let there be given the following linear programming problem

$$F(\mathbf{x}) = 3x_1 + 2x_2 \rightarrow \min$$

$$x_1 + 4x_2 \leq 72 \qquad \textbf{(2.13)}$$

$$6x_1 + 3x_2 \geq 120 \qquad \textbf{(2.14)}$$

$$x_1 + x_2 \leq 32 \qquad \textbf{(2.15)}$$

$$x_1 + x_2 \geq 26 \qquad \textbf{(2.16)}$$

$$x_1 \geq 10 \qquad \textbf{(2.17)}$$

$$x_2 \geq 8 \qquad \textbf{(2.18)}$$

Adding to the constraints set the inequality

$$3x_1 + 2x_2 \leq \varsigma$$

one obtains the polyhedron E(ς) with the measure $\Psi(\varsigma)$. After the minimization of Ψ (ς) according to the algorithm one obtains $\varsigma^* = 66$. For this value ς^* constraints (**2.13**), (**2.15**), (**2.17**) and (**2.18**) are proved to be excessive. Then solving the linear system

$$6x_1 + 3x_2 = 120$$

$$x_1 + x_2 = 26$$

one finds $x_1 = 14$, $x_2 = 12$.

2.3. A general scheme for the discrete Ψ-transform method

Let us proceed in the following way.

A general scheme for the discrete Ψ-transform method is implemented as an algorithm involving the following steps:

- generation of a set of random points in the feasible area D;
- determination of the forecast estimates (**2.3**), (**2.4**) on the basis of the set obtained;
- local optimization of the discrete variables vector of the problem (**2.1**), (**2.2**).

The following were chosen as weighting functions:

$$\rho(\mathbf{x},\varsigma)=\left|\varsigma-F(\mathbf{x})\right|^{n},\quad \beta(\mathbf{x},\varsigma)=\left[\frac{\varsigma}{F(\mathbf{x})}\right]^{n}$$

It is known that a discrete analog of the set measure is set cardinality. Hence, we have a discrete representation of Ψ-function as follows

$$\Psi(\varsigma)=\frac{|E(\varsigma)|}{|D|} \tag{2.19}$$

where $E(\varsigma)=\{\mathbf{x}\mid F(\mathbf{x})\le\varsigma,\mathbf{x}\in D\}$ and a representation of a general Ψ-function in the form

$$\Psi(\varsigma)=\frac{\sum\limits_{x\in D}\rho(\mathbf{x},\varsigma)\cdot\Theta(\mathbf{x},\varsigma)}{|D|}\ . \tag{2.20}$$

The first step of a general scheme for the Φ-transform method consists of the generation of the randomly distributed points { $\mathbf{x}^{\mathbf{i}}\in$ D, $\mathrm{i}\in\overline{1,N_1}$ }. There are a number of possible ways of doing this:

- on the basis of the penalty functions;
- by means of an approximation to a feasible area D by a simpler area (for example, if D is a hyperparallelepiped).

Find $F_{\min}=\min\limits_{i}F(x^{(i)})$ and interval $\left[\breve{\varsigma},\hat{\varsigma}\right]$ where

$$\hat{\varsigma}=F_{\max};\breve{\varsigma}=F_{k},k=\min(\breve{k},\hat{k}),$$

$\breve{k}$ - the number of different values function F(**x**) takes in the area $\{x^{(i)} \in D, i = \overline{1, N_1}\}$;

$\hat{k}$ - a given number (in the calculations below k = 20).

$$\varsigma_m = \varsigma - (m-1) \cdot \Delta\varsigma, \tag{2.21}$$

$$\Delta\varsigma = \frac{1}{k} \cdot (F_{\max} - F_{\min}).$$

Additionally, the random points $x^{(i)} \in D, i = \overline{N_1 + 1, N}$ are generated.

At the second step one needs to define a prognostic value for the global extremum

$$\varsigma^* = \max\{\varsigma | \Psi(\varsigma) = 0\}$$

and corresponding values for vector **x**.

There are, for example, the two following approaches for finding a prognostic value ς^*. The first one is based on an approximate evaluation of the function

$$\Psi(\varsigma) = \frac{\sum\limits_{x \in D} \rho(\mathrm{x}, \varsigma) \cdot \Theta(\mathrm{x}, \varsigma)}{|D|}$$

at the points ς_ν, $\nu = \overline{1, k}$

$$\Psi(\varsigma_\nu) = \frac{1}{N} \cdot \sum_{i=1}^{N} |\varsigma_\nu - F(x^{(i)})|^m \cdot \Theta(\mathrm{x}, \varsigma_\nu) \tag{2.22}$$

and a further extrapolation of $\Psi(\varsigma)$ at the zero point, i.e.

$$\Psi(\varsigma) = 0 \ .$$

The second method is reduced to the determination of a confidence interval with the level p for ς^* and to the choice of a corresponding value

$$\varsigma^* = F_1 + (1-\frac{1}{d})\cdot\frac{F_2 - F_1}{1-\frac{1}{p}}$$

where F_1, F_2 are the smallest and the next smallest values of F(**x**) found during the generation of random points;

d is a parameter allowing the selection of the value ς^* from the confidence interval I(p) to be varied.

To find coordinates of the global optimum point ς^*, find components of the generalized gravity centers E(ς_ν), $\nu=\overline{1,k}$ on the basis of the next formula

$$\bar{x}_j(\varsigma_\nu) = \frac{\sum\limits_{i=1}^{N} x_j^{(i)}\cdot\left[\varsigma_\nu / F(\mathbf{x}^{(i)})\right]^m \cdot\Theta(\mathbf{x}^{(i)},\varsigma_\nu)}{\sum\limits_{i=1}^{N}\left[\varsigma_\nu / F(\mathbf{x}^{(i)})\right]^m \cdot\Theta(\mathbf{x}^{(i)},\varsigma_\nu)} \qquad \textbf{(2.23)}$$

where $x_j^{(i)}$ is the j-th coordinate of i-th random point, j=1,m, i=1,N. As a result of the approximation to functions $\bar{x}_j(\varsigma)$ on the basis of the set of calculated points **(2.23)**, one can obtain the optimal value $x_j^* = \bar{x}_j(\varsigma^*)$.

By definition, vector $\mathbf{x}^*\in R^n$ defines a gravity center of the set E(ς^*) of the optimal vectors for the problem **(2.1, 2.2)**. Thus, a vector $\mathbf{x}_g^* \in D$ of discrete variables may be obtained as

$$\mathbf{x}_g^* = \arg\min_{x\in D}\left\|\mathbf{x}^* - \mathbf{x}\right\|^2$$

where $\left\|\mathbf{x}\right\| = (\sum\limits_{j=1}^{m} x_j^2)^{\frac{1}{2}}$ is a norm of vector **x**.

Suppose now,

$$\langle \bar{\mathbf{x}}(\varsigma) \rangle = \arg\min_{x \in D} \|\bar{\mathbf{x}}(\varsigma) - \mathbf{x}\|^2 .$$

Thus, as a result of the *second step* of the algorithm, the prognostic value of the global extremum $\varsigma^*$and the corresponding vector $\mathbf{x}$ will be found. If

$$\hat{\varepsilon} = \left(\left[F(\mathbf{x}_j^*) - \varsigma^* \right] / \varsigma^* \right)_+ > 0$$

then there is an error of approximation which may be decreased by the variaton method, in the Ψ-transform plane. This results in obtaining the value of

$$\hat{\varsigma}^* = \arg\min_{\varsigma} F\left[\langle \bar{\mathbf{x}}(\varsigma) \rangle \right]$$

and the corresponding coordinates of the vector $\mathbf{x}$ providing a global extremum for the function F($\mathbf{x}$). The given vector $\hat{\mathbf{x}}_g^*$ is used as basic at the final step of the algorithm to perform a local optimization.

The procedure of local optimization consists of transforming the discrete value of every j-th component of the vector $\hat{\mathbf{x}}_g^*$ starting with $\hat{\mathbf{x}}_j^*$ until it remains within the allowable boundaries, provided other components are satisfied. If F($\mathbf{x}$) is a unimodal function then the discrete values $\hat{\mathbf{x}}_j^*$ should be subsequently modified until F($\mathbf{x}$) decreases. Suppose, $\mathbf{x}''$ provides F($\mathbf{x}''$) < F($\hat{\mathbf{x}}_g^*$). Then $\mathbf{x}''$ is used at the next iteration of the minimization procedure, etc.

2.3.1. Choosing the method for an estimation of the global minimum of ς^*

As was pointed out earlier, there are two approaches to prognosticate the value of ς^*. In accordance with the first approach, a set of points ς_υ has been approximated by algebraic polynomials, Chebyshev's polynomials and by analogical means. As a result, the value

$$\hat{\varsigma}^* = \arg\min_{\varsigma} F[\langle \bar{x}(\varsigma)\rangle]$$

could be found with corresponding vector $\hat{\mathbf{x}}_g^* = \left\langle \bar{\mathbf{x}}(\bar{\varsigma}^*)\right\rangle$ providing $F(\hat{\mathbf{x}}_g^*)$ is approaching its global minimum. To provide the improvement of F(**x**), a procedure of local optimization should be used which starts with a given vector $\hat{\mathbf{x}}_g^*$ and finishes with some new vector $\tilde{\mathbf{x}}_g^*$ of the local minimum of $F(\tilde{\mathbf{x}}_g^*)$.

To approximate the function $\Psi(\varsigma)$ we have used the algebraic polynomial $\sum\limits_{k=0}^{2} a_k \cdot \varsigma^k$ defined in the points $\varphi_v = \Psi(\varsigma_v), v = \overline{1,k}$. The coefficients a_0, a_1, a_2 have been determined on the basis of a linear regression technique, i.e. the values of a_0, a_1, a_2 were found from the equation

$$\sum_{v=1}^{k} \gamma_v [\Psi(\varsigma_v) - \sum_{k=0}^{2} a_k \varsigma_v^k]^2 \to \min \qquad \textbf{(2.24)}$$

with weighting coefficients γ_v such that

$$\gamma_v = \left|\Psi(\varsigma^*) - \Psi(\varsigma_v)\right|^{-m} = \left[\Psi(\varsigma_v)\right]^{-m}. \qquad \textbf{(2.25)}$$

The last equation is connected with an assumption, that the closer $\Psi(\varsigma_v)$ is situated to 0, the stronger is its influence on ς^* and, therefore, the larger its corresponding weight γ_v. On the basis of a_0, a_1, a_2 and roots ς_1, ς_2 of an approximating polynom we can find

$$\varsigma^* = \begin{cases} \max\{\varsigma_1, \varsigma_2\}, & a_2 \geq 0 \\ \min\{\varsigma_1, \varsigma_2\}, & a_2 < 0\ . \end{cases}$$

In the last two cases we performed an approximation of the reverse function Ψ^{-1} using set of points $\varsigma_v = \Psi^{-1}(\varphi_v), v = \overline{1,k}$,by means of an algebraic polynomial with order h, given in the form

$$\Psi^{-1}(\varphi) = \sum_{k=0}^{h} a_k \cdot P_k(\varphi)$$

where $P_k(\varphi)$ is a Chebyshev's polynomial of order k. We also used as an approximation, a cubic polynomial spline with q nodes given by coefficients b_k in the representation

$$\Psi^{-1}(\varphi) = \sum_{k=1}^{q} b_k \cdot Q_k(\varphi)$$

in the system of fundamental splines $Q_k(\varphi)$.

Coefficients $a_k, k = \overline{0,h};\ b_m, m = \overline{1,q}$ and the number q of nodes were defined on the basis of the mean risk minimization approach [32].

It has been established that all the considered methods for estimating ς^* are too imprecise. In order to obtain a more precise estimation, let us use different approach [33].

Theorem 2.4. [33]. If there exist positive constants r and χ such that

$$\lim_{\varsigma \to \varsigma^*} \frac{\Psi(\varsigma)}{\left(\varsigma - \varsigma^*\right)^{\frac{1}{r}}} = \chi$$

then the probability P $\{\varsigma^* \in [\breve{F}, F_1]\ \}$ assymptoticaly approaches the given value $0 < p < 1$ for

$$\breve{F} = F_1 - \frac{F_2 - F_1}{p^{-r} - 1},$$

$$F_1 = \min F(x^{(i)}), i = \overline{1,N},$$

$$F_2 = \min_i \left\{ F(x^{(i)}) \middle| F(x^{(i)} \neq F_1, i = \overline{1,N} \right\}.$$

Theorem 2.5. Let r=1 and $\rho(\mathbf{x},\varsigma) = |\varsigma-F(\mathbf{x})|$. Then

$$\lim_{\varsigma\to\varsigma^*}\frac{\Psi(\varsigma)}{\varsigma-\varsigma^*}=\frac{|E(\varsigma^*)|}{|D|}.$$

Proof. Let $\varphi(\varsigma) = \dfrac{\Psi(\varsigma)}{\varsigma-\varsigma^*}$ and assume that F(**x**) takes values $\varsigma^* < \varsigma_1 <\ldots< \varsigma_z$ from the set D. From the conditions of theorem 2.1 it follows that for any $\varsigma \in [\varsigma_l, \varsigma_{l+1}], l = \overline{1,z}$ and $\rho(\mathbf{x},\varsigma)=|\zeta-F(\mathbf{x})|$ it is true that

$$\varphi(\varsigma)=\frac{|E(\varsigma^*)|\cdot(\varsigma-\varsigma^*)+\sum\limits_{v=1}^{l}|E(\varsigma_v)|\cdot(\varsigma-\varsigma_v)}{|D|\cdot(\varsigma-\varsigma^*)}=$$

$$\frac{|E(\varsigma^*)|}{|D|}+\sum_{v=1}^{1}\frac{|E(\varsigma_v)|}{|D|}\cdot\frac{\varsigma-\varsigma_v}{\varsigma-\varsigma^*}. \qquad \mathbf{(2.26)}$$

It is not difficult to note that for any $\varepsilon > 0$ for which $\varsigma = \varsigma^*+\varepsilon < \varsigma_1$ the second addendum in (**2.26**) becomes zero and $\varphi(\varsigma)$ takes a constant value of $|E(\varsigma^*)|/|D|$.

Hence, to estimate ς^* precisely one needs to select the correct value from the interval $[\breve{F}, F_1]$, where $\breve{F} = F_1 - (F_2 - F_1)/(\frac{1}{p} - 1)$. Suppose, for example,

$$\varsigma^* = \breve{F} + (F_1 - \breve{F})/d \qquad \mathbf{(2.26a)}$$

which gives

$$\varsigma^* = F_1 + (1-\frac{1}{d})\cdot\frac{F_2 - F_1}{1-\frac{1}{p}}. \qquad \mathbf{(2.26b)}$$

Here, d enables us to control the correct choice of the value ς^* from interval $[\breve{F},F_1],1\leq d\leq m$. As follows from [33], the usage of the assymptotic formula $P\{\varsigma^* \in[\breve{F},F_1]\}=p$ from **theorem 2.4** is legitimate only in the case when F_1 and F_2 are close to the ς^*-value. Otherwise, the result may be too imprecise. For the problems with high dimensions it is advantageous to increase the probability p and parameter d which cause the shifting of ς^* to the lower bound of F.

2.3.2 Approximation of the function $\bar{x}_j(\varsigma)$

The problem of the approximation to the function $\bar{x}_j(\varsigma)$ on the set of points (**2.23**) is reduced to the determening of its value in the point ς^*. Obviously, the closer is the interval $[\breve{\varsigma},\hat{\varsigma}]$ to the point ς^*, the more precise is the value of $\bar{x}_j(\varsigma)$. Bearing in mind this consideration , let us modify the method of selecting an interval $[\breve{\varsigma},\hat{\varsigma}]$.

Having calculated $F_{\min}$, $F_{\max}$, $\Delta\varsigma$ on the basis of the set of points $\{x^{(i)} \in D, i=\overline{1,N}\}$ let us now define a set of values

$$\varsigma_u = F_{\max} -(u-1)\cdot\Delta\varsigma \tag{2.27}$$

where

$$v=\begin{cases} L, |E(\varsigma_{L-1})| \neq |E(\varsigma_L)| \\ \min\{u, |E(\varsigma_u)|=|E(\varsigma_{u+1})|; u=L-1,\dots,1\}, if\, |E(\varsigma_{L-1})|=|E(\varsigma_L)| \end{cases}$$

$$L=\max\{u, |E(\varsigma_u)|\neq 0, u=1,2,\dots\}$$

Then, from the points (**2.27**) let us choose the last k values ς_u setting up $\hat{\varsigma}=\varsigma_{v-k+1}, \breve{\varsigma}=\varsigma_v$. Find $\bar{x}_j(\varsigma)$ for every point $\varsigma_v \in[\breve{\varsigma},\hat{\varsigma}]$ according to the formula (**2.23**).

Some experiments have been conducted to estimate the accuracy of the approximation given by different methods, namely:

- linear regression methods (M1);

- exponential smoothing method (M2), and
- minimization of a mean structural risk method (M3).

In the M1-method the approximation is provided by polynomials of the form

$$\bar{x}_j(\varsigma)=\sum_{k=0}^{2} a_{jk}\cdot\varsigma^k. \qquad \textbf{(2.28)}$$

The points of the approximation have been choosen unevenly to provide the highest accuracy of $\bar{\mathrm{x}}_j(\varsigma)$. This is due to the supposition that the gravity centers $\bar{x}_j(\varsigma)\in \mathrm{R}^n$ of the corresponding sets E(ς_v) seriously affect the estimated value of $\bar{\mathrm{x}}_j(\varsigma)$. The most serious drawback of the given approach, however, is connected with the necessity to apply a more complicated technique in order to solve the relatively simple initial problem. Thus, it may occur that the given set of points $\bar{x}_j(\varsigma)$ is insufficient to provide the necessary accuracy of the approximation. From this viewpoint it may be more desirable to use the M3-based approach [32,34] oriented at the restricted statistical samples. The general scheme of this approach is the following. A fixed set G of linear functions of the form

$$G(\varsigma,a)=\sum_{k=1}^{h} a_k\cdot g_k(\varsigma)$$

should be selected with parameter vector **a** and linear functions $g_{k(}\varsigma)$. It is assumed that G is somehow ordered, for example by the following ordering

$G_1\subset G_2 \subset \ldots\subset G_m\subset G.$

For every subclass G_c, a least-squares estimator $\hat{a}$ (c) and an estimate J(c) for the upper creadibility level η of a mean risk I(c) are further found in such a way that they satisfy s inequalities:

$P\{I(c) < J(c)\} > 1-\eta,\ c = \overline{1,s},$

where $0 < n < 1$.

Then the best estimate $\hat{a}^*$ is being searched for among estimates $\tilde{a}$ (c),c = $\overline{1,s}$. This value $\hat{a}^*$ corresponds to the minimum value J(c), c = 1,s. As a result, an approximating function G(ς,$\hat{a}^*$) enables one to restore the value G(ς^*,$\hat{a}^*$) in the point ς^*.

An advantage of this approach is based on a compromise between complexity and accuracy of the computational procedure. We investigated the efficiency of the approach for two different classes of functions G:

- algebraic polynomials of the order h given by the coefficients a_{ij} in the representation

$$\vec{x}_j(\varsigma) = \sum_{k=0}^{h} a_{jk} \cdot P_k(\varsigma) \tag{2.29}$$

where $P_k(\varsigma)$ is Chebyshew's polynomial of the order k;

- cubic polynomial splines with q nodes given by coefficients b_{ij} in the representation

$$\bar{x}_j(\varsigma) = \sum_{k=1}^{q} b_{jk} \cdot Q_k(\varsigma) \tag{2.30}$$

where $Q_k(\varsigma)$,k=$\overline{1,q}$ is a system of fundamental splines.

The method has been proved to be effective only for the interpolation of ς^* in the interval $[\breve{\varsigma},\hat{\varsigma}] \supset \varsigma^*$ .In the case $\varsigma^* < \breve{\varsigma}$, however, one should solve the extrapolation problem, for example, on the basis of the exponential smoothing method [35]. According to this method the restored value $\bar{x}_j(\varsigma)$ at the point ς^* is defined by the values $\varsigma_1,\dots,\varsigma_k$ as follows

$$x_j^* = S_{j,k+1} + \frac{\varsigma_k - \varsigma^*}{\Delta\varsigma} \cdot R_{j,k+1} \tag{2.31}$$

where parameters $S_{j,k+1}, R_{j,k+1}$ are computed on the basis of the recursive scheme below:

$$S_{j,v+1} = \alpha \cdot \bar{x}_j(\varsigma_v) + (1-\alpha)\cdot(S_{jv} + R_{jv}); \qquad \textbf{(2.32)}$$

$$R_{j,v+1} = \beta \cdot (S_{j,v+1} - S_{j,v}) + (1-\beta)\cdot R_{j,v}, v = \overline{1,k}; \qquad \textbf{(2.33)}$$

$$S_{j1} = \bar{x}_j(\varsigma_1), R_{j1} = 0. \qquad \textbf{(2.34)}$$

Here, S_{jv} is an exponentially weighted mean value of the function $\bar{x}_j(\varsigma)$ for $\varsigma > \varsigma_v$;

R_{jv} is an estimate of a trend of the function $\bar{x}_j(\varsigma)$ at the point ς_v ;

$0 \le \alpha \le 1, 0 \le \beta \le 1$ are weighting coefficients with respect to the corresponding values $\bar{x}_j(\varsigma_v), v = \overline{1,k}$ which are choosen on the basis of the try-and-test principle [36,37].

All the approximation methods considered above were used in the control test on the basis of the Steinberg problem [**38**]. The highest accuracy was provided by the exponential smoothing technique in the case $|E(\varsigma_v)| = 1$. This result has been taken into account in the discrete Φ-transforming algorithm which is outlined below.

2.3.3. Discrete Ψ-transform method for F-indefinite problems

Summing up of all the results obtained gives an algorithm for solving discrete optimization problems on the basis of Ψ-transform technique.

Inputs: - problem (**2.1, 2.2**);

the numbers of the basic points N_1, N;

parameters $\hat{k}$, m, p, d.

Outputs: - an approximated value ς^* of a global minimum of the objective function and

vector $\dot{x}$ providing ς^*;

the value ε of possible error of approximation.

```
begin
    find a set of random points $\{\mathbf{x}^{(i)} \in D\}, i = \overline{1,N}$
    compute the values $F_{min}$, $F_{max}$, k, $\Delta\varsigma$;
        for i=1,2,...,$N_1$ do to find $\varsigma_u$ by (2.27);
for $i=N_1+1,\ldots,N$ do
    begin
        generate $\mathbf{x}^{(i)} \in D$;
        define $\varsigma_u$ by (2.27);
    end
for $v = 1,2,\ldots,k$ do $\varsigma_u = \varsigma_{u-k-v}$;
   $F_1 := \min \{F(\mathbf{x}^{(i)})|\ i=1,N \}$;
   $F_2 := \min \{F(\mathbf{x}^{(i)})\ |F(\mathbf{x}^{(i)} \neq F_1,\ i=1,N \}$;
    define $\varsigma^*$ by (2.26b);
    for $j = 1,2,..,n$ do
    for $n = 1,2,\ldots,k$ do  define $\bar{x}_j(\varsigma_v)$ by (2.23)
      call MES($\varsigma^*$);
      $\varepsilon := ([F(\mathbf{x}_g^*)-\varsigma^*]/\varsigma^*)_+$;
      if $\varepsilon > 0$ then begin
      $\mathbf{x}_g^* := \operatorname{argmin} \{F(\mathbf{x}_g)|\mathbf{x}_g = \mathrm{VAR}(\varsigma)\}$;
      call MPP($\mathbf{x}_g^*$);
      define $\varsigma^*$ by (2.26b);
      $\varepsilon := [F(\mathbf{x}^0)-\varsigma^*]/\varsigma^*)$;
      end
   else $\mathbf{x}^0 := \mathbf{x}_g^*$;
end
```

The algorithm uses 3 procedures: MES, VAR and MPP. The first realizes an exponential smoothing technique. Procedure VAR is defined as follows.

Inputs: $\alpha^*, \beta^*, \bar{x}_j(\varsigma_v)$, j = 1,n; v = 1,k, $\varsigma^*, \Delta\varsigma$.

Outputs: $\mathbf{x}_g \in$ D

```
procedure VAR($\varsigma^*$)
 begin $\alpha := \alpha^*, \beta = \beta^*$;
 for j = 1,2,...,n do
   begin
     $S_{j1} := \bar{\mathbf{x}}_j(\varsigma_1)$, $R_{j1} := 0$;
        for n= 1,2,...,k do
        compute $S_{j,v+1}$ and $R_{j,v+1}$    by (2.32-2.34);
        compute $\mathbf{x}_j^*$ by (2.31);
   end
 return(<$\mathbf{x}^*$>);
 end;
```

Procedure MES is the following.

Inputs: ς^*; F_{max}; $\bar{\mathbf{x}}_j(\varsigma)$,j=1,n;$\Delta\varsigma$

Outputs: $\mathbf{x}_g^* \in D; \alpha^*, \beta^*$

```
procedure MES( $\varsigma^*$ );
   begin
               $F^* := F_{max}$ ;
               $F_0 := F^*$ ;
               $\alpha' := 0.1$;
               $\alpha'' := 0.9$;
               $h_1 := 0.1$;
               $\beta' := 0.1$;
               $\beta'' := 0.9$;
               $h_2 := 0.1$;
               $\beta^* := 0.5$;
  M1: $\beta := \beta^*$
          for $\alpha = \alpha', \alpha'+h, ..., \alpha''$  do
          if $F^* < F_0$ then begin
               $F_0 := F^*$ ;
```

```
                    $\alpha' := \alpha^* - h_1$ ; $\alpha'' := \alpha^* + h_1$;
                    $\beta' = \beta^* - h_2$ ; $\beta'' := \beta^* + h_2$ ;
                    $h_1 := h_1 / 10$;   $h_2 := h_2 / 10$;
                     goto M1;
                      end
                  end
```

The definition of the procedure MPP is given for the case of integer variables.

Inputs: vector $\mathbf{x}_g^* \in D$

Outputs: An approximate solution $\mathbf{x}^0$ to the problem **(2.1,2.2);** F_1 ,F_2 .

```
   procedure MPP($\mathbf{x}_g^*$ )
      begin $\mathbf{x}^0 := \mathbf{x}_g^*$ ; $F_1, F_2 := F(x)$,
 M1:  for j=1,2...,n do
         begin
      $X_{j+} := \{\mathbf{x} | \mathbf{x} \in D, x_j := x_j^0 + 1, x_j^0 + 2, ...\}$;
      $X_{j-} := \{\mathbf{x} | \mathbf{x} \in D, x_j := x_j^0 - 1, x_j^0 - 2, ...\}$
      $X_j := X_{j+} \cup X_{j-}$ ;
         end

      $X^* :=$ argmin $\{F(\mathbf{x}) | \mathbf{x} \in \bigcup_{j=1}^{n} X_j\}$;
      $F_1^* := F(\mathbf{x}^*)$;

      $F_2^* :=$ min $\{F(\mathbf{x}) | \mathbf{x} \in \bigcup_{j=1}^{n} X_j, F(\mathbf{x}) \neq F_1^*\}$;
       if $F(\mathbf{x}^0) > F_1^*$ then
                              $\mathbf{x}^0 := \mathbf{x}^*$ ,
                              $F_1 := F_1^*$ ,
            $F_2 := F_2^*$ ,
        goto M1;
    end
```

Discrete Ψ-transform method has proved, however, not to be effective in the case when an objective function F(**x**) had more than one optimum point. This also was the case when one of local optimum was slightly different from the global one. Other strategies for finding the global optimum of an objective function F(.) are given in [39 - 41].These are based on dividing a set D into a number of subsets, one of which contains an optimal point. To sum up, let us list the main positive features of the discrete Ψ-transform method:

- a possibility to predict the value of the function F global optimum with a corresponding error of approximation;

- a possibility to use this method in the case of a non-analytical representation of an objective function (i.e. an objective function is given by the set of pairs $<x_i, y_i>$, y_i = = $F(x_i)$).

- an effective computational realization.

2.4. *An approximate solution to F-indefinite static optimization problems*

Let us apply the described technique based on the Ψ-transform method to the combinatorial and integer-variable optimization problems. It provides us with a good example of using a weak solving strategy for the problems known as NP-complete.

2.4.1. F-indefinite mixed integer programming problems

Consider the following problem formulation:

$$F(\mathbf{x}) \to \min \qquad (2.35)$$

$$\mathbf{x} = \{x_1, x_2, \ldots, x_{n1}, \ldots, x_n) \in D \qquad (2.36)$$

where $D = \{x|g_i(x) \leq b_i, i = 1,m;$

$x_j \in N, j=1,n_1 ; x_k \in R, k = n_1 +1,n\};$

$g_i : R^n \rightarrow R, \ i = 1,m;$

N is the set of integers and R is the set of real numbers. As earlier, we suppose that function $F(\mathbf{x})$ is strictly positive or strictly negative.

Let $\mathbf{x} = (\mathbf{y},\mathbf{z})$, where $\mathbf{y} = (y_1 ,y_2 ,...,y_{n_1})$ and $\mathbf{z}$ = = $(z_1 ,z_2 ,...z_{n-n1})$; $\mathbf{y}$ and $\mathbf{z}$ are subvectors of vector $\mathbf{x}$.

$D_y = \{\mathbf{z}|(\mathbf{y},\mathbf{z}) \in D, \mathbf{z} \in R^{n-n1} \}, \mathbf{y} \in N^{n1}$

$D_z = \{\mathbf{y}|(\mathbf{y},\mathbf{z}) \in D, \mathbf{z} \in D_y ,\mathbf{y} \in N^{n1} \};$

$E_y (\varsigma) = \{\mathbf{z}|F(\mathbf{y},\mathbf{z}) \leq \varsigma ,\mathbf{z} \in D_y \}, \mathbf{y} \in N^{n1} ;$

$E_z (\varsigma) = \{\mathbf{y}|F(\mathbf{y},\mathbf{z}) \leq \varsigma , \mathbf{y} \in D_z \}, \mathbf{z} \in R^{n-n1};$

$Q_y(\mathbf{z},\varsigma)$ and $Q_z(\mathbf{y},\varsigma)$ are characterization functions of the corresponding sets $E_y(\varsigma)$ and $E_z (\varsigma)$;

N^{n1} - is the set of all $\mathbf{y}$ such that $y_j \in N$, j = 1,n .

In the case when 0 < n1 < n ,the Ψ-transform method has some peculiarities. Thus, $\Psi(\varsigma)$ takes the form

$$\overline{\Psi}(\varsigma) = \frac{\sum\limits_{y \in D_z D_y} \int \rho[(\mathrm{y},\mathrm{z}),\varsigma] \cdot \Theta_y(\mathrm{z},\varsigma) \partial z}{\sum\limits_{y \in D_z D_y} \int \Theta_y(\mathrm{z},F_{\max}) \partial z} \qquad \textbf{(2.37)}$$

where $F_{\max} = \max\limits_{x \in D} F(\mathbf{x}).$

Theorem 2.5. Let $F(\mathbf{y},\mathbf{z}) \in L_p(D_y)$ for every fixed $\mathbf{y} \in N^{n1}$ .Then for all $\varsigma \leq \varsigma^*$, function $\overline{\Psi}(\varsigma)$ in (**2.37**) has a null value and , for $\varsigma > \varsigma^*$, it is a continuous strictly increasing function.

To find the coordinates of the global optimum of the function F(x) under the condition 0< n1 < n ,let us compute the generalized values of the mass centers

$\bar{\mathbf{y}}(\varsigma) \in R^{n1}$and $\bar{\mathbf{z}}(\varsigma) \in R^{n-n1}$ of the corresponding sets $E_Z(\varsigma)$ and $E_y(\varsigma)$ from (**2.37**, **2.38**):

$$\bar{y}_j(\varsigma) = \frac{\sum\limits_{y \in D_z} y_j \cdot \int\limits_{D_y} \beta[(\mathbf{y},\mathbf{z}),\varsigma] \cdot \Theta_z(\mathbf{y},\varsigma) \partial z}{\sum\limits_{y \in D_z} \int\limits_{D_y} \beta[(\mathbf{y},\mathbf{z}),\varsigma] \cdot \Theta_z(\mathbf{y},\varsigma) \partial z} \tag{2.38}$$

$$\bar{z}_k(\varsigma) = \frac{\sum\limits_{y \in D_z} z_k \cdot \int\limits_{D_z} \beta[(\mathbf{y},\mathbf{z}),\varsigma] \cdot \Theta_y(\mathbf{y},\varsigma) \partial z}{\sum\limits_{y \in D_z} \int\limits_{D_z} \beta[(\mathbf{y},\mathbf{z}),\varsigma] \cdot \Theta_y(\mathbf{y},\varsigma) \partial z} . \tag{2.39}$$

Theorem 2.6. Let F(**y**,**z**) ∈ $L_p(D_\mathbf{y})$ for every fixed **y** ∈ N^{n1}, | E (ς)| = 1 for every **z** ∈ R^{n-n1} and the set $E_y(\varsigma^*)$ be compact for every **y** ∈ N^{n1} . Then $\mathbf{y}^* = \bar{\mathbf{y}}(\varsigma^*)$and $\mathbf{z}^* = \lim\limits_{\varsigma \to \varsigma^*} \bar{\mathbf{z}}(\varsigma)$ where ($\mathbf{y}^*$, $\mathbf{z}^*$) = $\mathbf{x}^* \in$ Argmin F(**x**). This theorem directly follows from theorem 2.1.

The general scheme of the discrete Ψ-transform method may be realized as a kind of integer programming algorithm:

- generating the set of random points X = {$\mathbf{y}^{(i)}$,$\mathbf{z}^{(ki)}$ } ∈ D,
- obtaining estimates ς^* and $\mathbf{x}^*$ for the set X; correcting the solution obtained;
- optimizing locally the objective function in the $\mathbf{x}^*$-area.

Functions ρ (**x**,ς) = $|\varsigma\text{-F}(\mathbf{x})|^S$ and $\beta(\mathbf{x},\varsigma) = [\frac{\varsigma}{F(\mathbf{x})}]^S$, S > 0 are selected as before.

The following theorems are used to find an estimate of ς^* in (**2.9**).

Theorem 2.7. [33]. Let n_1 = 0 and F(**x**) have the single global minimum at the point $\mathbf{x}^* = (\ \mathbf{y}^*,\ \mathbf{z}^*) \in$ D. Let F(**x**) have the first and the second derivative F'(x) and F''(**x**) respectively. Suppose that the Hessian H($\mathbf{x}^*$) is not degenerate. Then the limit

$$\lim_{\varsigma\to\varsigma^*}\frac{\mu[E(\varsigma)]}{(\varsigma-\varsigma^*)^{\frac{n}{2}}}$$

exists and has a positive value.

Theorem 2.8. Let 0≤ n_1 < n, F(x) has the single global minimum at the point $x^*=(y^*, z^*)$, $z^* \in R^{n-n1}$, 0≤ n_1 < n, $z^*\in Dy^*$. Suppose, that F(y,z) has the first and the second derivative at the point z^* and the Hessian $H(z^*)$ is not degenerate. Then the limit

$$\lim_{\varsigma\to\varsigma^*}\frac{\Psi(\varsigma)}{(\varsigma-\varsigma^*)^{\frac{n-n1}{2}}}$$

exists and has a positive value.

Theorem 2.9. Let n = n_1 , ρ(x,z) = | ς - F(x) | Then

$$\lim_{\varsigma\to\varsigma^*}\frac{\Psi(\varsigma)}{\varsigma-\varsigma^*}=\frac{|E(\varsigma^*)|}{|D|}.$$

It follows from the above theorems that we should use an estimate ς^* which is defined as shown below:

$$\varsigma^*=\begin{cases}F_1-(1-\frac{1}{d})\cdot\dfrac{F_2-F_1}{p^{-\frac{2}{n-n1}}-1},0\le n1<n\\ F_1-(1-\frac{1}{d})\cdot\dfrac{F_2-F_1}{p^{-1}-1},n1=n.\end{cases} \qquad \textbf{(2.40)}$$

To compute the approximate coordinates of the point x^* we suggest using the following formulae:

$$\overline{y}_j(\varsigma_\nu)=\frac{\sum_{i=1}^{M}\sum_{k_i=1}^{M_i} y_j^{(i)}\cdot[\varsigma_\nu/F(\mathbf{y}^{(i)},\mathbf{z}^{(k_i)})]^s\cdot\Theta_{z^{(k_i)}}(\mathbf{y}^{(i)},\varsigma_\nu)}{\sum_{i=1}^{M}\sum_{k_i=1}^{M_i}[\varsigma_\nu/F(\mathbf{y}^{(i)},\mathbf{z}^{(k_i)})]^s\cdot\Theta_{z^{(k_i)}}(y^{(i)},\varsigma_\nu)},$$

$$\overline{z}_k(\varsigma_\nu)=\frac{\sum_{i=1}^{M}\sum_{k_i=1}^{M_i} z_k^{(k_i)}\cdot[\varsigma_\nu/F(\mathbf{y}^{(i)},\mathbf{z}^{(k_i)})]^s\cdot\Theta_{y^{(i)}}(\mathbf{z}^{(k_i)},\varsigma_\nu)}{\sum_{i=1}^{M}\sum_{k_i=1}^{M_i}[\varsigma_\nu/F(\mathbf{y}^{(i)},\mathbf{z}^{(k_i)})]^s\cdot\Theta_{y^{(i)}}(\mathbf{z}^{(k_i)},\varsigma_\nu)} \qquad \textbf{(2.41)}$$

j=1,n1; k=1,n-n1.

These formulae enable one to compute a series of numbers for further extrapolation at the point ς^* . The points **y'** ∊ R^{n1} and **z'** ∊ R^{n-n1} define a mass center **x** = (**y'**, **z'**) of the set E(ς^*)=Argmin F(**x**). Vector **x*** may be found as

$$\mathbf{x}^* = Arg\min_{x\in D} \|\mathbf{x}'-\mathbf{x}\|^2$$

where ||**x**|| is a norm of the vector **x**.

Assume,

$$<\mathbf{x}^{*\prime}(\varsigma)> = <\mathbf{x}'(\varsigma)>=Arg\min_{x\in D}\|\overline{\mathbf{x}}(\varsigma)-\mathbf{x}\|^2.$$

If the error ε of approximation is positive then in order to reduce ε one should use the method of variations in the Ψ-transform plane which produces more exact values $\varepsilon^{*\prime}$ and $\mathbf{x}^{*\prime}$. These values are used at the final step of the considered scheme - in a local optimization procedure consisting of the subsequent definition of each component j(k) of vector $\mathbf{x}^{*\prime}$ = ($\mathbf{y}^{*\prime}$, $\mathbf{z}^{*\prime}$). The procedure commences with components y_j and z_k and continues until the values generated remain within set D.

At every c-th iteration ,the procedure forms two subsets of points Y_{j+} and Y_{j-} such that

$$Y_{j-}=\{\mathbf{y}\left|\begin{array}{l}(\mathbf{y},\mathbf{z}^{(c-1)})\in D,\\ y_i=y_i^{(c-1)},(i\neq j);\\ y_j=y_j^{(c-1)}-1,y_j^{(c-1)}-2,\ldots;\}\end{array}\right.$$

$$Y_{j+}=\{\mathbf{y}\left|\begin{array}{l}(\mathbf{y},\mathbf{z}^{(c-1)})\in D,\\ y_i=y_i^{(c-1)},i\neq j;\\ y_j=y_j^{(c-1)}+1,y_j^{(c-1)}+2,\ldots;\};\\ j=\overline{1,n1}\end{array}\right.$$

(at the first iteration $\mathbf{y}^{(0)}=\mathbf{y}^{*'}$, $\mathbf{z}^{(0)}=\mathbf{z}^{*'}$).

Find $\mathbf{y}^{(c)} = \mathrm{argmin}\{F(\mathbf{y}, \mathbf{z}^{(c-1)})\}$. If $F(\mathbf{y}^{(c)}, \mathbf{z}^{(c-1)}) < F(\mathbf{y}^{(c-1)}, \mathbf{z}^{(c-1)})$ then $\mathbf{y}^{(c)}$ is used at the next (c + 1)-th iteration.

The optimization of the subvector $\mathbf{z}^{*'}$ is quite different, however. At the 1-st iteration find

$$Z_k^L=\inf(Z_k^{(0)}),\ Z_k^U=\sup(Z_k^{(0)}),i=\overline{1,n-n1}$$

where k is k-th component of subvector $\mathbf{z}^{*'}$;

$$Z_k^{(0)}=\{z_k\left|(\mathbf{y}^{(1)},\mathbf{z})\in D,z_i=z_i^{*'}\}\right..$$

At the c-th iteration form finite subsets Z_{k+} and Z_{k-} , k = 1,n-n1 where

$$Z_{k-}=\{\mathbf{z}\left|\begin{array}{l}(\mathbf{y}^{(c)},\mathbf{z})\in D,\ \ z_i=z_i^{(c-1)},\ \ (i\neq k)\\ z_k=z_k^{(c-1)}-h_k;\ z_k^{(c-1)}-2h_k,\ldots,z_k^L\end{array}\right.$$

$$Z_{k+}=\{\mathbf{z}\left|\begin{array}{l}(y^{(c)},\mathbf{z})\in D,\ \ z_i=z_i^{(c-1)},\ \ (i\neq k)\\ z_k=z_k^{(c-1)}+h_k;\ \ z_k^{(c-1)}+2h_k,\ldots,z_k^U\end{array}\right.$$

$$h_k=\frac{z_k^U-z_k^L}{4},\qquad\qquad i=\overline{1,n-n1}.$$

Choose $\mathbf{z}^{(c)} = \operatorname{argmin}\ \{\ F(\mathbf{y}^{(c)}, \mathbf{z}_{\min}); F(\mathbf{y}^{(c)}, \mathbf{z}^{(c-1)}\ \}$, where $\mathbf{z}_{\min} = \operatorname{argmin}\ \{\ F(\mathbf{y}^{(c)}, \mathbf{z}) | \mathbf{z} \in \bigcup_{k=1}^{n-n1} Z_k$. Choose also an index l corresponding to vector $\mathbf{z}^{(c)}$ and such that $\mathbf{z}^{(c)} \in Z_l = Z_{l+} \cup Z_{l-}$.

If there exists an index k = 1, n-n1 such that $h_k > \varepsilon$ (ε is a given positive number) then $\mathbf{z}^{(c)}$ is used at the next (c + 1)-th iteration with new upper and lower bounds

$$Z_l^L = Z_l^{(c)} - h_l,$$
$$Z_l^U = Z_l^{(c)} + h_l.$$

If Z_l contains more than one member, then assume

$$Z_l^L = \min\{z_l^{(c)}\} - h_l; \qquad Z_l^U = \max\{z_l^{(c)}\} + h_l \ .$$

If $h \leq \varepsilon$ for given positive number ε then $z_l^{(c)}$ = const.

As it is observed from the above considerations, the scheme is divided into two stages:

- mapping $\tilde{\mathbf{x}}$ to the set D and
- truncating the integer variables to the nearest allowable values in D.

The first stage is connected with the problem in the following form:

$$f(\mathbf{x}) = \|\tilde{\mathbf{x}} - \mathbf{x}\|^2 \to \min \tag{2.42}$$

$$\mathbf{Ax} \leq \mathbf{b} \tag{2.43}$$

where $\mathbf{A}_{mn}$ is a matrix of real coefficients; $\mathbf{b} \in R^n$.

Due to the pequliarities of the problem (**2.42**, **2.43**) there is a number of effective methods, for example see [41,42]. Rewrite the above formulation in the form

$$f(\mathbf{x}) = \frac{1}{2}(\mathbf{x},\mathbf{x}) - (\tilde{\mathbf{x}},\mathbf{x}) \to \min \tag{2.44}$$

$$\mathbf{Ax} \leq \mathbf{b} \tag{2.45}$$

where $(\tilde{\mathbf{x}}, \mathbf{x})$ is a scalar product of two vectors in R^n Obviously, (**2.44, 2.45**) are equivalent to (**2.42**, **2.43**) respectively and the dual representation may be written as

$\Psi(\mathbf{u})=1/2\cdot(\mathbf{AA}^t,\mathbf{u}) + (\mathbf{b}\text{-}\mathbf{A}\tilde{\mathbf{x}}\ ,\mathbf{u}) \to \min$

$u_i \geq 0,\ i\in[1,m]$

or in general form:

$$\Psi(\mathbf{u})=1/2\cdot(\mathbf{Du},\mathbf{u}) + (\mathbf{c}\ ,\mathbf{u}) \to \min \qquad (\mathbf{2.46})$$

$$u_i \geq 0,\ i\in[1,m] \qquad (\mathbf{2.47})$$

where $\mathbf{D} = \mathbf{AA}^t$ is a square matrix of an order m; $\mathbf{c}\ = \mathbf{b}\text{-}\mathbf{A}\tilde{\mathbf{x}}\ ,\mathbf{c},\mathbf{u}\in\mathbf{R}^m$ is a dual vector.

It is seen that problem (**2.46**, **2.47**) has a simpler representation as it requires only the non-negativeness of variables u_i , $i\in[1,m]$.

Assertion 2.6. In the problem (**2.44**, **2.45**) let solution set $\mathbf{X} = \{\ \mathbf{x}\ |\mathbf{Ax} \leq \mathbf{b}\ \}$ be non-empty. Then problem (**2.46**, **2.47**) has an optimal solution $\mathbf{u}^*$ such that

$\tilde{\mathbf{x}}\ {}^* = \tilde{\mathbf{x}} - \mathbf{u} \bullet \mathbf{A}$

is , in its own turn ,the solution of the problem (**2.44**, **2.45**).

Definition. Point **u** providing a solution of the problem

$\Psi(\mathbf{u}) \to \min$

$u_i \geq 0,\ i\in I = [1,m]$ (I may be empty)

is called a characteristic point.

Assertion 2.7. The set of all characteristic points of the problem (**2.46**, **2.47**) is finite.

The problem (**2.46**, **2.47**) is solved by two procedures. The first one builds a characteristic point and the second one tests if this characteristic point is optimal or not. If not , then it looks for another characteristic point **u** with a lower value $\Psi(\mathbf{u})$. This scheme guarantees only decreasing values of the function $\Psi(\mathbf{u})$ and by virtue of assertion 2.7, should terminate with the optimal solution after a finite number of iterations.

If vector $\tilde{x}^*=(\tilde{y}^*,\tilde{z}^*)$ is obtained , then at the second stage the subvector $\tilde{y}^*$ should be truncated to the nearest integer vector from D. It should be noted that such a truncation does not guarantee an optimality of the resulting vector $x^*=<\bar{x}(\varsigma^*)>$ but in most practical applications the result should be considered acceptable.

To sum up the above considerations let us provide an algorithm to solve (**2.42**, **2.43**).

```
Inputs: $\tilde{x} \in R^n$; problem (2.42, 2.43); set D.
Outputs point $x^* \in D$

procedure PRK($\tilde{x}$ )
        begin
        for  i = 1, 2, ..., m  do
                  $u_i' := 0$;
          M:= { 1, 2, ..., m };
          D:= $\mathbf{A}\,\mathbf{A}^t$ ,
          $\mathbf{c} := \mathbf{b} - \mathbf{A}\tilde{x}$ ;
M4:  $\mathbf{r} := \nabla\Psi(\mathbf{u}')$;
comment: $\mathbf{r} = (r_1 , r_2 , ..., r_m )$, $\nabla\Psi(\mathbf{u}') = \mathbf{D}\mathbf{u}' + \mathbf{c}$;
   $I(\mathbf{u}') := \{ i \mid u_i' = 0, i \in M \}$;
            for all $i \in I(\mathbf{u}')$ do
              if $r_i < 0$
              then goto M1;
               for  all $i \notin I(\mathbf{u}')$
           do if $r_i \neq 0$ then go to M1
           $\mathbf{u}^* := \mathbf{u}'$ ; goto M2;
M1: for all $i \in I(\mathbf{u}')$ do
      $h_i' := -\min\{0;\mathrm{sign}( r_i )\}$;
   for all $i \notin I(\mathbf{u}')$ do
      $h_i' := -\mathrm{sign}( r_i )$;
```

```
        $\alpha = -(\nabla\Psi(\mathbf{u}'),\mathbf{h}') / (D\mathbf{h}', \mathbf{h}')$;
        $\mathbf{h}^0 := \mathbf{h}' + \alpha\mathbf{h}'$;
M3:         for all $i \in M$ do
            if $u_i^0 = 0$ then
             begin
      $M := M \setminus i$;
  delete i- th row and i- th column from D;
  delete i- th element from $\mathbf{c}$;
            end
       $\mathbf{h}^0 := -\nabla\Psi(\mathbf{u}^0)$;
M5: $\alpha_0 := -(\nabla\Psi(\mathbf{u}^0), \mathbf{h}^0) / (D\mathbf{h}^0, \mathbf{h}^0)$;
        for all $i \in M$ do $u_i^0 := u_i^0 + \alpha_0\cdot h_i^0$;
         for all $i \in M$ do
         if $u_i^0 < 0$ then
  begin
        for all $u_i^0 < 0, i \in M$ do $u_i^0 := 0$;
              goto M3;
  end
         if $\nabla\Psi(\mathbf{u}^0) = 0$ then $\mathbf{u}' := \mathbf{u}^0$,
         goto M4;
         $\beta := (\nabla\Psi(\mathbf{u}^0), D\mathbf{h}^0) / (D\mathbf{h}^0, \mathbf{h}^0)$;
          $\mathbf{h}^0 := -\nabla\Psi(\mathbf{u}^0) + \beta\bullet\mathbf{h}^0$;
        goto M5;
M2:     $\tilde{\mathbf{x}}* := \tilde{\mathbf{x}} - \mathbf{u}*$

comment: $\tilde{\mathbf{x}}^* = (\tilde{\mathbf{y}}^*, \tilde{\mathbf{z}}^*)$;
M7: for i = 1, 2, ..., $n_1$ do
        if $|\lfloor\tilde{y}^*\rfloor - \tilde{y}^*| \le |\lceil\tilde{y}^*\rceil - \tilde{y}^*|$ then
        $\mathbf{y}^* := \lfloor\tilde{\mathbf{y}}^*\rfloor$ else $\mathbf{y}^* := \lceil\tilde{\mathbf{y}}^*\rceil$
comment: $\lfloor y\rfloor$ is the greatest integer which is less than y;
```

```
            $\lceil y \rceil$ is the least integer which is greater than y;
        if $x^* = ( y^* , \tilde{z}^* ) \in D$ then
        return ( $x^*$ );
          for i = 1, 2, ..., $n_1$ do
          if $\tilde{y}_i^* - \tilde{y}_i \neq 0$ then $\lambda := \tilde{y}_i^* - \tilde{y}_i$,
          goto M6;
M6: if $\lambda < 0$ then $\gamma := -1/\lambda$ ;
  else $\gamma := 1/\lambda$
        for i = 1, 2, ..., $n_1$ do
          goto M7;
          $\tilde{y}_i^* := \tilde{y}_i^* + \lambda \cdot \gamma$
            end
```

It should be noted that the above algorithm may be ineffective when point $\tilde{x}^* = (\tilde{y}^*, \tilde{z}^*)$ coincides with the vertex of cone and point $x_* = (y_* , \tilde{z}^*) \in D$ is not the nearest one in respect to the point $x_* = (y_* , \tilde{z}^*) \in D$.

Combining this result with that obtained earlier we have the following algorithm for solving partly integer linear programming problems:

Inputs: problem (**2.35**, **2.36**); the set $M_i^{(1)}$, i = 1, M_1 of basic points, M_i ; i = 1, M ; k, s, p, d.

Outputs: ς^*, x_0 - an approximate solution to the problem (**2.35**, **2.36**); the value ε of error.

```
    begin
    generate the set of random points $\{(y^{(i)}, z^{(ki)}) \in D$
    compute $F_{min}$ , $F_{max}$ , k, $\Delta\zeta$;
    $N_1 := \prod_{i=1}^{M_1} M_i^{(1)}$ ;
    for i:=1, 2, ...,$N_1$
```

```
     do
      define $\zeta\cdot\mathbf{u}$ by (**2.27**);
  for i:= $M_1$ + 1,...,M do
  for $K_i$:= $M_i^{(1)}$+1,..,$M_i$
      do
 begin
    generate {($\mathbf{y}^{(i)}$,$\mathbf{z}^{(ki)}$) ∈ D
    define $\varsigma$ by (**2.27**);
  end
   for $\upsilon$ = 1, 2, ..., k do
    $\varsigma_v$:= $\varsigma_{v\text{-}k+v}$
  $F_1$ := min {F ($\mathbf{y}^{(i)}$,$\mathbf{z}^{(ki)}$ ) $k_i$ = 1, $M_i$ ,
 $F_2$ := min {F ($\mathbf{y}^{(i)}$,$\mathbf{z}^{(ki)}$ ) | F ($\mathbf{y}^{(i)}$,$\mathbf{z}^{(ki)}$ )≠ $F_1$
 define $\varsigma^*$ by (**2.40**)
 for $\upsilon$ = 1, 2, ..., k do
   begin
     for j = 1,2,...,$n_1$
     do
define $\bar{y}_j(\varsigma_v)$ by (**2.41**)
       for j = 1,2,...,n-$n_1$
       do
define $\bar{z}_k(\varsigma_v)$ by (**2.41**);
  end
     call MES ($\varsigma^*$);
     ε := $([F(x_g^*)-\varsigma^*]/\varsigma^*)_+$;
if ε > 0 then begin
$\hat{\mathbf{x}}^*$:=$\underset{\varsigma}{\mathrm{argmin}}\{F(\mathbf{x})|\mathbf{x}=VAR\,(\varsigma)\})$ };
call MPP ($\hat{\mathbf{x}}^*$ );
```

```
      define $\varsigma^*$ by (2.40)
       $\varepsilon$:=$[F(\mathbf{x}^0)-\varsigma^*]/\varsigma^*$
    end
       else $\mathbf{x}^0$:=$\mathbf{x}_g^*$;
  end
```

Procedures MES, VAR and HPP have the same meaning as earlier. Procedure VAR takes now the following form:

Inputs: α^* , β^* ; $\bar{y}_j(\varsigma_v), \bar{z}_k(\varsigma_v)$, $j = 1,n_1$,

$k = 1, n\text{-}n_1$, $v = 1, k$; ς^* , $\Delta\varsigma$

Outputs: $\mathbf{x}^* \in D$.

```
procedure VAR ( $\varsigma^*$ )
            begin
           $\alpha$:= $\alpha^*$
           $\beta$:=$\beta^*$
             for j = 1,2,...,n do
              begin
               $R_{j1}$:=0;
              if j ≤ $n_1$ then $S_{j1}$ := $\bar{y}_j(\varsigma_1)$
      else $S_{j1}$ := $\bar{z}_{j-n_1}(\varsigma_1)$
            for v = 1, 2, ..., k do
        compute $S_{j,v+1}$ , $R_{j,v+1}$ by (2.32-2.34)
        if j ≤ $n_1$ then
        define $\tilde{y}_j$ by (2.41),
       $\tilde{x}_j$:=$\tilde{y}_j$ else
       define $\tilde{z}_{j-n_1}$ by (2.41) and
       set $\tilde{x}_j$:=$\tilde{z}_{j-n_1}$ ;
      end
      $\mathbf{x}_* := \mathrm{PRK}(\tilde{x})$;
    return ( $\mathbf{x}_*$ );
  end
```

Procedure MPP has the following form

Inputs: vector $\hat{\mathbf{x}}^* = (\hat{\mathbf{y}}^*, \hat{\mathbf{z}}^*) \in D$, ε.

Outputs: an approximate solution $\mathbf{x}^0 = (\mathbf{y}^0, \mathbf{z}^0)$; F_1 F_2 .

```
 procedure MPP ( $\hat{\mathbf{x}}^*$ )
    begin
    $\mathbf{y}^0 := \hat{\mathbf{y}}^*$, $\mathbf{z}^0 := \hat{\mathbf{z}}^*$ ;
    $F_1 := F(\mathbf{x}^0)$, $F_2 := F_1$ ;
M1:      for  j=1,2,...,$n_1$
       do   begin
       $Y_{j+} := \{ \mathbf{y} | (\mathbf{y}, \mathbf{z}^0) \in D, y_i = y_i^0,\ i \neq j,\ i = \overline{1,n_1}$;
       $y_j = y_j^0 + 1,\ y_j^0 + 2, \ldots \}$
       $Y_{j-} := \{ \mathbf{y} | (\mathbf{y}, \mathbf{z}^0) \in D, y_i = y_i^0,\ i \neq j,\ i = \overline{1,n_1}$
       $y_j = y_j^0 - 1,\ y_j^0 - 2, \ldots \}$
       $Y_j := Y_{j+} \cup Y_{j-}$ ;
       end

    $\mathbf{y}_* := \arg\min \{F(\mathbf{y}, \mathbf{z}^0), \mathbf{y} \in \bigcup_{j=1}^{n_1} Y_j \}$;
       $F_1^* := F(\mathbf{y}_*, \mathbf{z}^0)$;
       $F_2^* := \min\{F(\mathbf{y}, \mathbf{z}^0) \mid \mathbf{y} \in \bigcup_{j=1}^{n_1} Y_j,\ F(y, z^0) \neq F_1^* \}$
       if $F(\mathbf{y}^0, \mathbf{z}^0) > F_1^*$ then $\mathbf{y}^0 := \mathbf{y}_*$ ,
       $F_1 := F_1^*$ , $F_2 := F_2^*$ ,
         goto M1;
         ind := 1;
       for  $k=1,2,\dots,n-n_1$  do
       begin
      $Z_k := \{ z_k \mid (\mathbf{y}^0, \mathbf{z}) \in D,\ z_i = z_i^0,\ i=1,n-n_1,\ ( i □\neq k )\}$;
      $z_k^L := \inf Z_k,\ z_k^U := \sup Z_k$;
      $h := (z_k^U - z_k^L)/4$;

       end
M3:    for $k = 1,2,\dots,n-n_1$
        do
         begin
         $Z_{k+} := \{ \mathbf{z} \mid (\mathbf{y}^0, \mathbf{z}) \in D,\ z_i = z_i^0,\ i=1,n-n_1\ (i\neq k)$;
         $z_k = z_k^0 + h_k, \dots, z_k^U \}$;
         $Z_{k-} := \{ \mathbf{z} \mid (\mathbf{y}^0, \mathbf{z}) \in D,\ z_i = z_i^0,\ i=1,n-n_1\ (i\neq k)$;
         $z_k = z_k^0 - h_k, \dots, z_k^L$
          $Z_k := Z_{k+} \cup Z_{k-}$;
          end

          $z^* := \operatorname{argmin} \{F(\mathbf{y}^0, \mathbf{z}) \mid \mathbf{z} \in \bigcup_{k=1}^{n-n_1} Z_k \}$;
          $F_1^* := F(\mathbf{y}^0, \mathbf{z}^*)$;

          $F_2^* := \min \{F(\mathbf{y}^0, \mathbf{z}) \mid \mathbf{z} \in \bigcup_{k=1}^{n-n_1} Z_k,\ F(\mathbf{y}^0, \mathbf{z}) \neq F_1^* \}$;
    if $F(\mathbf{y}^0, \mathbf{z}^0) > F_1^*$ then ind := 0, $\mathbf{z}^0 := \mathbf{z}^*$, $F_1 := F_1^*$, $F_2 := F_2^*$; prh := 0;
          for   $k=1,2,\dots,n-n_1$  do
            if $h_k > \varepsilon$ then  prh := 1  goto M2;
M2:           for $k=1,2,\dots,n-n_1$  do
                if $\mathbf{z}^0 \in Z_k$ then
               begin
                $l := k$;
                  if $h_l \le \varepsilon$  then
```

```
                    $z_l^L := z_l^0,\ z_l^U := z_l^0,$ goto M4   else
                      begin
                      $L := \{ z_l \mid F(\mathbf{z}) = F(\mathbf{z}^0),\ \mathbf{z} \in Z_l \};$
                      $z_l^L$ := inf L-$h_l$;
                      $z_l^U$ := sup L+$h_l$ ;
                      $h_l := (z_l^U - z_l^L)/4;$
                      goto M3;
                      end
                  end
M4:              if prh = 1 then
                      begin
                      for k=1,2,...,n-$n_1$ do
                      if $h_k \le \varepsilon$ then
                      $z_k^L := z_k^0,\ z_k^U := z_k^0$   else
                          begin
                 $z_k^L := z_k^0 - h_k$
                 $z_k^U := z_k^0 + h_k$
                 $h_k := (\dfrac{z_k^U - z_k^L)}{4}$
                 end
                 goto M3
              end
              if  ind = 0 then goto M1;
             end
```

2.4.2. Application of the discrete Ψ-transform method to permutation problems

Let P be a set of permutations $P = \{ p_1, p_2, \dots, p_n \}$ defined for the discrete finite set $M = \{1,2,\dots,n\}$. Consider the problem in the following form:

$$F(P) \rightarrow \min \tag{2.48}$$

$$P = \{ p_1 , p_2 , \ldots , p_n \}. \tag{2.49}$$

As earlier, it is supposed that F(p) is a strictly positive (negative) function. Usually, all constraints in the problem (**2.48,2.49**) are given by a precedence relation..$\succ$, e.g. $p_i \succ p_j$ entails i < j. The general scheme of the solving procedure in this case is the same as before:

- generating the number of random permutations;
- estimating values ς* and **x***;
- correcting an approximation error ε and
- optimizating locally the basic permutation **P*** .

Let an estimate of ς* of the objective function F(p) and the values of $x_j(\varsigma_\upsilon)$, $\upsilon = 1,k$ of corresponding functions $x_j(\varsigma)$ be known. Taking into account that vector **x*** = **x**(ς*), represents the mass center of the set E(ς*) of optimal permutations , find one of these by solving the following problem

$$\varphi(p) = \sum_{j=1}^{n} (x_j^* - p_j^2).$$

Previously give

Lemma 2.1 [43]. Let $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_n$ and $\beta_1, \beta_2, \ldots, \beta_n$ are two series of numbers.

Then permutation p = $\{p_1, p_2, \ldots, p_n\}$ provides a maximum of the sum $\sum_{j=1}^{n} \alpha_j \cdot \beta_{p_j}$ if and only if $\beta_{p1} \geq \beta_{p2} \geq, \ldots, \geq \beta_{pn}$.

Assertion 2.8. Let $\mathbf{x}_1^* \geq \mathbf{x}_2^* \geq \ldots \geq \mathbf{x}_m^*$. Permutation $p^* = \{p_1^*, p_2^*, \ldots, p_n^*\}$ minimizes function φ(P) if and only if $p_1^* \geq p_2^* \geq \ldots \geq p_n^*$.

Set $< \overline{x}(\varsigma) >= \underset{p \in P}{\operatorname{argmin}} \sum_{j=1}^{n} (\overline{x}_j(\varsigma) - p_j)^2$. When P coincides with the set of all m! possible permutations, definition of p* = < $\overline{x}(\varsigma^*)$)> is reduced to the minimizing function φ(p) in accordance with **assertion 2.8**. As earlier, p* is an approximate solution and is further used to find a corrected example $p^{**} = \text{argmin}\ \{F(\ p\ ) \mid p = =<\bar{x}(\varsigma)>\}$. In its turn, p^{**} represents a basic permutation used in the local optimization phase. We suggest using a sequential search based on the mutual transposition scheme with the complexity $O(\ m^2\ )$. If this scheme results in a better solution p' than the current basic permutation, then p' becomes a basic one and is used in the following iterations. We confine our considerations to the MPP-procedure which realizes a sequential search scheme.

Inputs: basic permutation p^* ;

Outputs: improved permutation p^{**}, F_1, F_2

```
          procedure MPP (p* )
          begin
            $p^{**} := p^*,\ F_1 := F(p^*),\ F_2 := F_1$ ;
M1:         $p := p^{**}$ ;
            for all $k \in M$ do
          begin
              $j_0 := p_k^0$ ;
              for $j = j_0 + 1, j_0 + 2, \ldots, n$   do
            begin
            $p_k := j$;
              for all $l \in M$ do
                if $p_l^0 = j$ then goto M2;
M2:     $p_l := j_0$ ;
      if $p \in P$ then
        begin
        if $F_1 > F(p)$ then $F_1 := F(p),\ p^* := p$;
        if $F_1 < F(p)$ and $F(p) < F_2$ then $F_2 := F(p)$;
         end
          $p_l := j$;
      end
```

```
            $p_k := j_0$
        end
            if $F(p^0) > F_1$ then $p^0 := p^*$, goto M1;
    end
```

2.5. Conclusion

The chapter has dealt with the discrete optimization problems with infinite sets of solving operators. It has demonstrated a weak strategy (called Ψ-transform method) for this class of problems. Our next task is to consider a class of problems with unknown sets of e.s.os.

This Page Intentionally Left Blank

Chapter 3

WEAK METHODS AND HEURISTIC REASONING

Abstract

This chapter deals with weak methods such as algorithmic procedures and heuristic principles. One can consider a heuristic principle(s) as a generator of a family of algorithms. We shall consider the general strategies and the solving principles which have high practical efficiency. As weak methods and heuristic reasoning form one of the main parts of a computer-aided problem solving system, they should be given special attention from the theoretical and practical viewpoints.

3.1. Specific features of solving tasks by weak methods

The following features are common for solving technologies based on weak methods:

(1) the task model may be partly indefinite;

(2) the proof of solution correctness may be absent and

(3) the solution may not be optimal.

Partial vagueness of a problem may be connected with insufficient knowledge about the task domain or its natural intricacy. We shall distinguish between the following basic weak strategies:

- restricted and directed "try-and-test" policy;
- cutting and exclusion;
- heuristics;
- induction.

In particular, a general strategy of building logical inference in a rule-based expert system is a mixed case of a restricted try-and-test strategy, heuristic reasoning and a cutting policy.

A principle of exclusion (cutting) represents one of the effective means of providing a reduction of the search area. If a solution does not have a feature P, then all variants which imply P or follow from P shoud be excluded (however, the latter may lead to losing a solution).

The most interesting realization of the cutting principle is associated with making suppositions.

The following statement is of this kind:

"removing a condition from the problem model generates alternatives and, vice versa, making a supposition (imposing a condition) on the problem domain leads to a reduction of the number of alternatives".

Let F be a set of given conditions defined on the task domain, and f be an additional set of conditions, that is $\{F \cup f\}$ represents the condition set for some particular problem with known solution R. This solution is applicable to the initial problem, in the case of $F \vdash f$, and may be adopted as basic if

$$\neg(F \vdash f) \& \neg(F \vdash \overline{f})$$

for a current state of the problem-related knowledge.

The basic solution may not be an optimal one, but does not contradict the problem's specification. An effective cutting policy is based on the consideration of mutually incompatible suppositions f and $\overline{f}$.

Let F be a set of problem related conditions. If an assumption of f leads to a contradiction, i.e. $\{ F, f \} \vdash \square$, then $\overline{f}$ is automatically adopted and vice versa.

A practically acceptable way of making assumptions is that associated with adopting the supposition f which is most probably false (it is the background of reductio ad absurdum reasoning).

Heuristics. In its own right a heuristic represents an empiric modus with rather high practical efficiency. An important part of an expert knowledge base consists of heuristic rules. Heuristics are also used for choosing rules in rule-based expert systems.

Induction. We have, in a way, discussed the essential features of induction earlier. The following inductive rules of Mill may serve as basic examples of inductive reasoning.

Rule 1. If n ≥ 2 cases have the single common feature, then this feature is a causa (or consequence) of a given phenomenon.

Denoting by A,B,C... the causae of some phenomenon and by a,b,c... its features we may write rule 1 schematically as follows

$$\begin{array}{l} A,B,C \mapsto a,b,c \\ A,D,E \mapsto a,d,e \\ \hline \quad A \mapsto a \end{array}$$

where symbol ↦ stands for relation " to be causa of ...".

Rule 2. If the case where given feature "a" is present, and the case where "a" is absent are identical in the remainder, and the given phenomenon arises in the first case and does not arise in the second, then "a" may be considered a causa (consequence) of the given phenomenon, i.e.

$$\begin{array}{c} A,B,C \mapsto a,b,c \\ B,C \mapsto b,c \\ \hline A \mapsto a \end{array} .$$

Rule 3. If a given part of the reasons is known to cause a definite set of consequences then the remaining part of the reasons is a causa of the remaining consequences, i.e.

$$\begin{array}{c} A,B,C \mapsto a,b,c \\ A \mapsto a \\ B \mapsto b \\ \hline C \mapsto c \end{array} .$$

Rule 4. Each phenomenon varying coherently with another phenomenon is either the causa or the consequence of that second phenomenon.

3.2. Control of the solving process

A control scheme for the solving process is either formalized in a model specification or represents an external part of the model. A good example of the latter possibility is the nondeterministic (AND - OR) - graph. Consider the case when the solution plan should be found for such a graph taking the following preliminaries. Let $x_1,x_2,\ldots,x_N$ be the objects in some area and for each object x_i, the corresponding procedure (s) is (are) known which calculates x_i. Denote by φ_i $(x_{i1},x_{i2},\ldots,x_{it})$ the procedure which defines x_i and takes the objects $x_{i1},\ldots,$ x_{it} as input parameters. In their own way the objects x_{ij} are calculated by other procedures φ_{ij} . Thus, we can write that

$$x_i = \varphi_i(x_{i1}, x_{i2}, \ldots, x_{it})$$

with a suitable graphical interpretation shown in Fig 3.1. The double arc in Fig 3.1. corresponds to the conjunction of the input parameters of the same procedure for the object x. If there is more than one procedure φ for the object x then we have an example of an OR-vertex for that object x (Fig. 3.2).

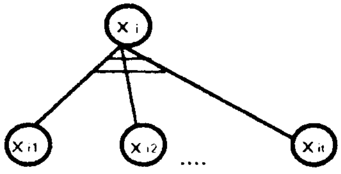


Fig. 3.1

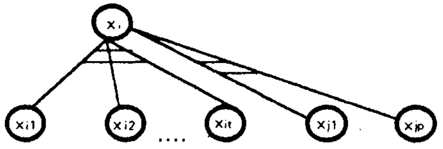


Fig. 3.2.

Let there be given procedures

$$\varphi_1^{n1}, \varphi_2^{n2}, \ldots, \varphi_k^{nk}$$

with the numbers of arguments n_1, n_2,..., n_k respectively.

Define the state S_i of the system as a set of objects which are determined at step i and let S_0 be an initial and S_e be a final state. The state S_e will be denoted by

$$S_e = < x_{e1}, x_{e2}, \ldots, x_{ez}, * >,$$

where an asterisk stands for any (may be empty) subset of objects, and objects $x_{e1}, x_{e2}, \ldots, x_{ez}$ are to be found in S_e .

One may consider a procedure φ_k $(x_{k1}, x_{k2}, \ldots, x_{km})$ to be valid in state S_j if

$$(x_{k1}, x_{k2}, \ldots, x_{km}) \subseteq S_j$$

or, more briefly,

$$S_j \mapsto \varphi_k^m .$$

As it can now be seen the problem consists in finding an ordered set C = $<\varphi_{i0}, \varphi_{i1}, \ldots,$ φiz > which provides a mapping

$$S_0 \overset{C}{\rightarrow} S_e$$

and every procedure φ_{it} is valid in state S_t from which φ_{it} starts.

Let us refer to an example. We shall assume that the designation

$$\varphi_i \;< x_{i1}, x_{i2}, \ldots, x_{in} | x_{in+1}, x_{in+2}, \ldots, x_{im} >$$

is used to distinguish between the subset of objects $x_{i1}, x_{i2}, \ldots, x_{in}$ which are inputs for the procedure φ_i and the subset of objects $x_{in+1}, x_{in+2}, \ldots, x_{im}$ which are the outputs of φ_i. Let

$$\varphi_1 : < x_1, x_3 | x_4 >$$

$$\varphi_2 : < \quad | x_1, x_2, x_5 >$$

$$\varphi_3 : < x_1, x_3 | x_4, x_6 >$$

$$\varphi_4 : < x_1, x_5 | x_7, x_9 >$$

$$\varphi_5 : < x_2, x_7 | x_4 >$$

$$\varphi_6 : < x_1, x_3, x_4 | x_6, x_8 >$$

$\varphi_7 :< x_3, x_6 \,|\, x_5, x_7 >$

$\varphi_8 :< x_5, x_6 \,|\, x_9 >.$

The corresponding (AND-OR) graph is shown in Fig.3.3. The nodes of the graph represent the objects x_i ($i = \overline{1,8}$) and the arcs are labeled in such a way that every arc connecting nodes x_i and x_j and ending in the node x_j is associated with the symbols of the procedures which use x_i as an input parameter and x_j as an output parameter. Thus, the arc $(\overline{x_1, x_4})$ is labeled with φ_1, φ_3 because φ_1 and φ_3 both take $x_1$as input and x_4 as output parameters.

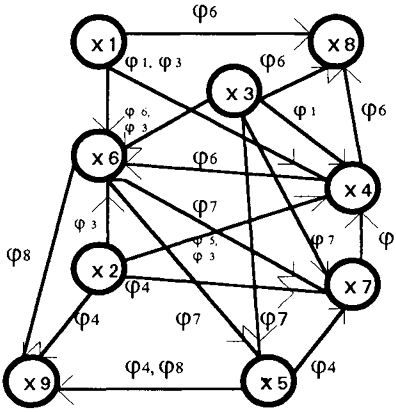


Fig. 3.3.

Let

$S_0 =< x_1, x_3, x_5 >$

and

$S_e =< x_7, x_9, * >.$

The whole procedure for finding a sequence C for the mapping

$S_0 \xrightarrow{C} S_e$

is performed in two stages.

Stage 1 (normalization of an initial AND-OR-graph).

The following normalizing operations are to be performed in arbitrary order as far as is possible:

(O_1) All labels of the functions with one or more input arguments deleted are to be removed from the graph;

(O_2) All input arcs of the nodes from S_0 should be deleted;

(O_3) if $x_i \notin S_e$ and x_i has no output arcs then x_i should be deleted with all its arcs;

(O_4) if x is an alternative node then it has to be deleted with all its arcs.

A node x_i is called alternative if it satisfies the following definition:

(i) $x_i \notin S_0 \cup S_e$.

(ii) Let x and y be connected by the arc $(\overline{x,y})$. Then the node x is an alternative node if for every node y ($x \neq y$) there is a node z ($z \neq x$) such that ($\overline{z,y}$) is the arc connecting nodes z and y and

$$L(\overline{z,y}) \setminus L(\overline{x,y}) \neq \varnothing,$$

where L ($\overline{a,b}$) is the set of labels for the arc ($\overline{a,b}$) and "\" is a set difference operation.

(iii) Deletion of x_i leads to the situation in which each vertex x_k from S_e remains attainable from S_0.

From Fig.3.3. one can establish that the node x_2 is an alternative node. Indeed, x_2 is connected with x_4 by the arc (x_2,x_4) with $L(x_2,x_4) = \{\varphi_3,\varphi_5\}$. Other node $z = x_3$ is connected by the arc (x_3,x_4) with the node x_4 and L(x_3, x_4) = { φ_1 }. It follows, further, that

$$\{\varphi_1\} \setminus \{\varphi_3,\varphi_5\} = \{\varphi_1\} \neq \varnothing;$$

(O_5) If some procedure φ_i has lost (by virtue of the above operations) all its output arguments then one should delete the identifier " φ_i " from the arcs of the labeling sets of the graph. If this results in losing all the identifiers for some arc, then this arc should be deleted;

(O_6) If an arc ending in the node x and containing identifier φ in its labeling set has been deleted then the identifier φ has to be removed from all the labeling sets assigned to the arcs incoming to x.

Let us return to example in Fig. 3.3. Remove node x_8 by virtue of operation O_3. Remove input arcs of the node $x_5 \in S_0$ and output arcs of the node $x_7 \in S_e$ (the resulting graph is shown in Fig. 3.4). We may now remove node x_2 as it is an alternative node (Fig. 3.5). Node x_4 in graph in Fig 3.5 is also alternative.

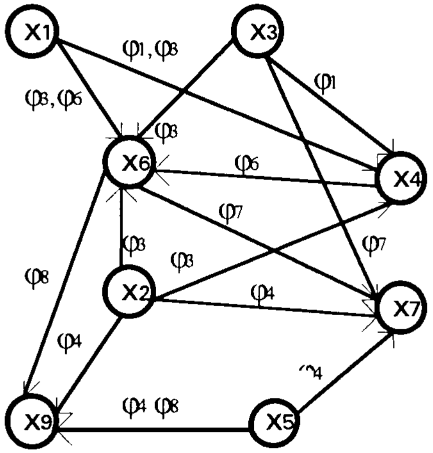


Fig.3.4.

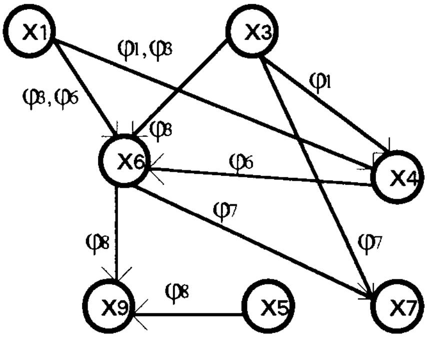


Fig.3.5.

Removing x_4 with all its arcs results in the graph shown in Fig. 3.6.

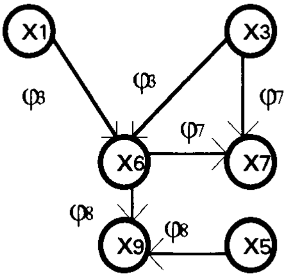


Fig.3.6.

Stage 2 (Dividing a normalized graph into levels).

This stage consists of the leveling of a normalized graph by levels $L_0, L_1, \dots, L_k$. L_0 contains the nodes having no input arcs. L_i contains the nodes x_l (not belonging to the levels $L_{i-1}, L_{i-2}, \dots, L_0$) which are calculated by some procedure φ_z valid in state $S_i = L_0 \cup L_1 \cup \dots \cup L_{i-1}$, i.e

$$S_i \mapsto \varphi_z.$$

Thus, from Fig. 3.6 one can directly derive the following definitions

$$L_0 = \{x_1, x_3, x_5\},$$
$$L_1 = \{x_6\},$$
$$L_2 = \{x_7, x_9\}.$$

Now every object x_k belonging to the level L_i ($i \neq 0$) is replaced by a procedure identifier φ_j if φ_j calculates x_k. Forbiding the same procedures identifier which is presented more than once in each level we obtain that

$$L_1' = \{\varphi_3\},$$
$$L_2' = \{\varphi_7, \varphi_8\}.$$

Finally, the resulting ordered set C is formed in such a way that procedure φ_t stands in C to the left of φ_q if and only if $\varphi_t \in L_n$, $\varphi_q \in L_m$, $n < m$; if $n = m$ then the mutual disposition of φ_t and φ_q in C may be arbitrary. This rule enables one to obtain, for instance, the following resulting set : $C = < \varphi_3, \varphi_7, \varphi_8 >$.

Let us conclude this section by considering the inference strategy controlled by data. Since the control structure is not explicitly defined in such models then one needs to use some special mechanisms:

- nondeterministic alternative choosing;
- backtracking;
- unification algorithm.

The mechanism of nondeterministic alternative choosing is based on a so-called heuristic evaluating function introduced by N. Nilsson. This is a subject of the next section.

3.3. Models of heuristic-based solution searching

Heuristic searching is based on the representation of the states of a problem by a tree with a root-node corresponding to the initial state of the problem. We shall denote by $\Gamma(x)$ for every node x all its direct succesors. Every node $y \in \Gamma(x)$ will also be called a child of the node x.

Select some goal node(s) corresponding to the final state.

Now we can formulate our task as follows. Let there be given initial (S_0) and final (S_e) states and the set of operators φ_j mapping each state into the others. As before, we are interested in the operating chain

$C = < \varphi_{i1}, \varphi_{i2}, \ldots \varphi_{it} >$

providing the mapping $S_0 \xrightarrow{C} S_e$ and corresponding to some routine $<S_0, S_1, \ldots, S_e>$ in the tree, beginning at the root and ending at the node S_e , such that

$S_i \in \Gamma(S_{i-1})$.

The known strategies of a search in the tree represent some kind of heuristic try-and-test disciplines. The most significant among them are the "Depth-First " discipline, cutting the search-area method, and heuristic evaluation function-based algorithms.

3.3.1.Depth-First discipline

Define a level of a node x in a search - tree on the basis of the following scheme:

(i) Root-node has a 0-level;

(ii) A level of a node x which is not a root-node is equal to a level of a node y ($x \in \Gamma(y)$) added with 1.

In "Depth-First" strategy, at every step the node is selected and opened with its maximum level value (to open a node x means to find $\Gamma(x)$). The whole procedure is the following [5]:

(1) The root-node is placed in the list which is called OPENED.

(2) If the OPENED-list is empty, than we have a general failure, otherwise go to next step.

(3) In OPENED-list find a node with a maximum level value and move this node to another list which is called CLOSED.

(4) If a level of the node α has a maximum allowable value then go to step (2) or else go to next step.

(5) Open the node α and place every node $\beta \in \Gamma(\alpha)$ in the OPENED-list if β does not belong either to OPENED-list or to the CLOSED-list.

(6) If $\Gamma(\alpha)$ contains a goal node then stop or else go to step (2).

To illustrate this strategy let us refer to an example in Fig. 3.7.

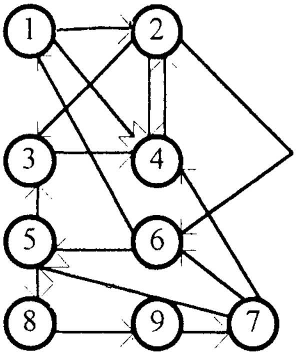


Fig.3.7.

Suppose, it is required to find a routine connecting nodes 1 and 7.The initial node 1 is the only member of the OPENED-list, i.e.,

$$OPENED = \{1\}$$

Further we find:

$$\Gamma(1) = \{2^{(1)}, 4^{(1)}\},$$

where the levels of the corresponding nodes are pointed out in round brackets. Assume that

$$OPENED = \{2^{(1)}, 4^{(1)}\},$$

$CLOSED = \{1^{(0)}\}$.

Choose node 2 to be opened:

$\Gamma(2) = \{3,4,6\}$.

Since node 4 has already been included in the OPENED-list then set

$OPENED = \{4^{(1)}, 3^{(2)}, 6^{(2)}\}$.
$CLOSED = \{1^{(0)}, 2^{(1)}\}$.
$\Gamma(3) = \{4\}$.

Node 4 is already included in the OPENED-list and, therefore, it remains in this list:

$OPENED = \{4^{(1)}, 6^{(2)}\}; CLOSED = \{1^{(0)}, 2^{(1)}, 3^{(2)}\}$.

Now open node 6 :

$\Gamma(6) = \{1,5\}$

and set

$OPENED = \{4^{(1)}, 5^{(3)}\}; CLOSED = \{1^{(0)}, 2^{(1)}, 3^{(2)}, 6^{(2)}\}$.

By analogy with the above steps, find that

$\Gamma(5) = \{3,8\}$,
$OPENED = \{4^{(1)}, 8^{(4)}\}$,
$CLOSED = \{1^{(0)}, 2^{(1)}, 3^{(2)}, 5^{(3)}, 6^{(2)}\}$
$\Gamma(8) = \{9\}$,
$OPENED = \{4^{(1)}, 9^{(5)}\}$,
$CLOSED = \{1^{(0)}, 2^{(1)}, 3^{(2)}, 5^{(3)}, 6^{(2)}, 8^{(4)}\}$.

Finally,

$\Gamma(9) = \{7\}$: the ending node is reached.

The whole routine we are interested in may be found in reverse order by means of the sets $\Gamma(\alpha)$. Thus, the ending node 7 belongs to the set $\Gamma(9)$, hence it follows that node 9 should be the last but one node in the routine.

We further find: 9 ∈ Γ(8), hence node 8 should precede node 9, etc. The resulting routine connecting nodes 1 and 7 in the initial graph is the following: <1, 2, 6, 5, 8, 9, 7>.

The algorithm we have considered above did not use any additional suppositions with respect to the nodes. In particular, if some additional information concerning the ending node is known, this would enable us to contract the number of opened nodes which could not belong to the resulting routine. This property is used by different cutting strategies embedded in searching procedures.

3.4. Try-and-test procedures with cutting

As an examples of these procedures we may point out the branches-and-boundaries method by Little et. al. and (α-β)- procedure by N. Nilsson.

3.4.1. Branches-and-bounds method

Let us consider the main ideas of this method. First of all, this method is connected with some general scheme of making alternatives and their evaluation. This scheme may be generally represented as shown in Fig. 3.8.

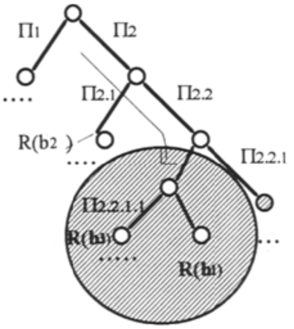


Fig. 3.8.

Nodes in Fig.3.8., as before, correspond to the states of the problem. There are only two directions from each node (this is, however, not a principal constraint). Each direction is assigned with indexed identifier П. Identifier R(b_i) represents some numeric value assigned to node b_i.

Commonly speaking, there are no restrictions on the number of levels in the solution tree. However, it is reasonable to try to keep its depth as small as possible. This consideration should be taken into account when choosing a supposition $П_m$: first of all one must try to prove a supposition which has the least plausability or (vice versa) try to refute a supposition which has maximum plausability (i.e., to adhere to the reductio ad absurdum principle). As it is very plausible that such a strategy leads to obtaining a contradiction then the corresponding alternative may be cut at the outset.

Let us look for the state b^* with minimum value of $R(b^*)$.Assume that we know some solution (state) b_x with current record R(b_x). It is clear then that each node b_i providing at best the value R(b_i) ≥ R (b_x) may be deleted (the corresponding part of search tree may be cut as shown in Fig. 3.8 by the shaded area).

For the sake of clarity consider the traveling salesman problem. Given a net with 4 nodes connected by the weighted arcs with the weights $C_{ij} \geq 0$ ($C_{ij} = \infty$), find the cycle passing through each node and having minimum total cost. We shall use the cost-matrix C = [C_{ij}] shown in Fig. 3.9.a,b.

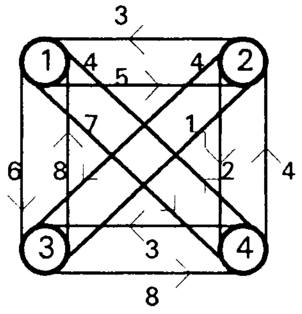


Fig.3.9 a

	1	2	3	4
1	∞	5	6	7
2	3	∞	4	2
3	2	1	∞	8
4	4	4	3	∞

Fig.3.9 b

To apply the branches-and-bounds method to this problem one has to take into account two basic ideas:

A1) it is required to select the only element in each row (column) of matrix C with the minimum total sum in such a way that these elements form a cycle. This last requirement is essential, for example, the elements $C_{12}, C_{23}, C_{31}, C_{44}$ do not form a cycle.

A2) if the same positive number D is subtracted from each element in the same row (column) then the total sum of elements forming an optimal cycle for the traveling salesman problem would decrease by the value of D.

Let us make use of the last property.

Find the minimal elements h_j in every column in the initial cost-matrix and then subtract them from the elements of the corresponding columns. Then find the minimal elements h_i in every row and subtract them from the elements of the corresponding rows. This gives us the matrix of Fig. 3.10 from which one can obtain the value $\hat{C}^*$ for the minimal total resulting cycle cost, i.e.,

$$\hat{C}^* = \sum_{i=1}^{4} h_i + \sum_{j=1}^{4} h_j = 11.$$

There are two alternative suppositions for every element in the matrix in Fig. 3.10, namely, one can assume, for example, that the arc $(\overline{3,4})$ belongs to an optimal cycle C* and vice versa, i.e., that $(\overline{3,4}) \notin C^*$.

	1	2	3	4	hi
1	∞	1	0	2	3
2	1	∞	1	0	0
3	0	0	∞	3	0
4	2	3	0	∞	0
hj	2	1	3	2	

Fig.3.10

If the arc $(\overline{3,4})$ really belongs to C^* then set $C_{4,3} = \infty$ and delete row 3 and column 4 which results in obtaining the matrix shown in Fig. 3.11. Using rule A2) again we obtain the matrix shown in Fig. 3.12. For this latter matrix one can find $\hat{C}^*\left[\overline{3.,4}\right] = 3$. It means that the cost of an optimal cycle containing the arc $(\overline{3,4})$ may be expressed as follows:

$$\hat{C}^* + \tilde{C}_{3,4} + \hat{C}^*\left[\left(\overline{3,4}\right)\right] = 11+3+3=17,$$

where $\tilde{C}_{3,4}$ is defined from the matrix in Fig. 3.12.

	1	2	3
1	∞	1	0
2	1	∞	1
4	2	3	∞
hj	1	1	0

Fig 3.11.

	1	2	3	hi
1	∞	0	0	0
2	0	∞	1	0
4	0	1	∞	1
hj	1	1	0	

Fig. 3.12

At the same time, a cost of the cycle $C = \left\langle\left(\overline{1,2}\right),\left(\overline{2,4}\right),\left(\overline{4,3}\right),\left(\overline{3,1}\right)\right\rangle$ is equal to 5+2+3+2=12 which allows us to draw a conclusion that arc $(\overline{3,4})$ does not belong to the optimal cycle. Hence, set $C_{3,4} = \infty$.

Suppose now that arc $(\overline{1,4})$ belongs to C^*. Omitting all the details we find that total cost of the optimal cycle containing this arc is equal to 13. Consequently, this arc does not belong to C^* and $C_{3,4} = \infty$. After these cosiderations, we see that there remains only one acceptable element in the column 4 of the matrix shown in Fig. 3.9.b , namely - $C_{2,4}$ =2. It is therefore correct that $(\overline{2,4}) \in C^*$.Hence, one should delete row 2 and column 4 with the result shown in Fig. 3.13. and $\hat{C}^*$=12.

Assume, that $(\overline{1,3}) \notin C^*$ From this supposition one can automatically derive that $(\overline{4,3}) \in C^*$. Delete row 4 and column 3 (see Fig. 3.14).

	1	2	3	hi
1	∞	1	0	0
3	0	1	∞	0
4	2	3	∞	0
hj	0	1	0	

Fig. 3.13.

	1	2
1	∞	1
3	0	1

Fig. 3.14

For this case it is required that $(\overline{3,1}) \in C^*$ and an optimal cycle has a total cost $\hat{C}^*=12$.

$$C^* = \left\langle (\overline{2,4}),(\overline{4,3}),(\overline{3,1}),(\overline{1,2}) \right\rangle.$$

Continuing by analogy, we obtain a search tree with the optimal cycle found with a total cost $\hat{C}^*=12$. The productivity of the branches-and-bounds method significantly depends on the accuracy of the estimation of the bounds it provides.

3.4.2. A searching strategy based on a heuristic evaluation function

The use of a heuristic evaluation function was proposed by N. Nilsson. Let us denote by f(n) the value of a heuristic evaluation function for the node n of a search tree. f(n) estimates the cost of an optimal path connecting the root-node of a tree, the given node n, and one of the goal nodes.

Bearing in mind that our goal is to find a route between the root-node and goal-node of a search tree, an algorithm to reach the goal is given by the following.

(1) Put root-node s into the list OPENED and find f(s).

(2) If OPENED $=\varnothing$ then stop algorithm with general failure; or else go to the next step.

(3) Select a node x in OPENED-list with minimal value of f (x).

(4) If x is a goal-node then stop algorithm (the goal is reached) or else find $\Gamma(x)$. For every node $y \in \Gamma(x)$ find f(y). If a node y does not belong to the list OPENED then the node y should be put in this list. If y is already in the OPENED-list then y is assigned with the minimum value f(y) selected from the given values of the heuristic evaluation function for that node.

(5) Go to step (2).

This algorithm is known as an algorithm A^*.

It is not difficult to conclude that the correct selection of the type of heuristic evaluation function forms the core of the whole approach. The function f in algorithm A^* has the form of

$$f(n) = g(n) + h(n),$$

where $g(n)$ represents the cost of an optimal path connecting the root-node to the given node n; $h(n)$ is the cost of an optimal path connecting given node n to the goal-node.

In addition, let $\hat{g}(n)$ be an estimation of the function $g(n)$,and $\hat{h}(n)$ in respect to the function $h(n)$, and $\hat{f}(n)$ in respect to the function $f(n)$, i.e.

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n).$$

From the definition of $g(n)$ we have

$$\hat{g}(n) \geq g(n).$$

It should be noted that the definition of $\hat{g}(n)$ does not generally cause any difficulties. The case of $\hat{h}(n)$, however, is quiet different. There are some theoretical considerations relevant to the definition of the function $\hat{h}(n)$.

Lemma 3.1 [5]. If for each n $\hat{h}(n) \leq h(n)$ then at any time before algorithm A* terminates there is an opened node n' in every optimal path P from root-node s to the goal-node such that $f(n') \leq f(s)$.

Proof. By definition,

$$\hat{f}(n') = \hat{g}(n') + \hat{h}(n').$$

Since n' belongs to the optimal path P then $\hat{g}(n') = g(n')$ and $\hat{f}(n') \leq f(n')$ for we have adopted that $\hat{h}(n') \leq h(n)$.

Note that for every pair of nodes x_1 and x_2 belonging to an optimal path P it follows that

$$f(x_1) = f(x_2) = f(s).$$

Indeed, let x_1 precede x_2 in P. Then

$$f(x1) = g(x1) + j(x1, x2) + h(x2),$$

where $j(x_1,x_2)$ is a cost of optimal path from x_1 to x_2 .

It is evident, however, that

$h(x1) = j(x1, x2) + h(x2)$ and

$$g(x2) = j(x1, x2) + g(x1).$$

All the above necessarily leads to the desired conclusion that

$$\hat{f}(n') \leq f(s).$$

Theorem 3.1 [5]. If for every node n, the relation $\hat{h}(n) \leq h(n)$ holds and if the cost of every arc in a search tree is greater than some low positive number δ, then algorithm A^* terminates successfully with an optimal-cost path P connecting the root-node and the goal-node.

Proof. There are three different cases.

Case 1. The algorithm stops but goal-node is not reached. It means that the OPENED-list is empty and there are no nodes to be expanded. This case is possible if and only if there is no path connecting the root-node to the goal-node.

Case 2. The algorithm never stops. Such an issue is impossible if a problem has a finite number of states. Let the opposite be true, i.e., the search tree has a finite size but the algorithm never terminates. It implies that the OPENED-list is never empty, i.e. the same nodes will be put in it many times and the values of f(n) will subsequently decrease.

If some node is once again included in the OPENED-list then it means that a new path from the root-node to the goal-node is found. Since the total number of all different paths is restricted then the initial supposition for the case 2 is contradictory.

Case 3. An algorithm terminates at the goal node, but the path found is not optimal. Assume, that this is a goal t and $\hat{f}(t) = \hat{g}(t) \geq f(s)$. However, according to the lemma, just prior to the termination of the algorithm there should exist a node n' such that

$f(n') \leq f(s)$. It implies that not node t but node n' would be expanded first ,and this contradicts the suppositions of case 3.

It is said that a heuristic evaluation function h satisfies a monotonic constraint if for every pair of nodes x and y, such that

$$y \in \Gamma(x)$$

the relationship

$$h(x) \geq h(y) + c(x, y)$$

where c(x.y) is a cost of the arc $\overline{(x, y)}$, takes place.

Theorem 3.2. If function h satisfies a monotonic restriction then A^* is optimal.

Proof. Find

$$\delta_x = \max_{y \in \Gamma(x)} h(y) + c(x, y)$$

Then ,for every node x, suppose

$$\hat{h}(x) = \delta_x .$$

It is clear that $h(x) \geq \hat{h}(x)$ for all x. By virtue of theorem 3.1, A^* is optimal. Despite the fact that A^* finds a minimum-cost path connecting the root-node to the goal-node, this algorithm has an exponential complexity. In this connection all attempts to improve the computational characteristics of A^* are quite natural. Consider, for example, the approach of Ghallab & Allard [44]. They have suggested a heuristic algorithm A_ε which is faster than its counterpart. This algorithm realizes a "deep-first" searching strategy which prefersto expand those nodes belonging to the same path in search tree. It is assumed that node n is acceptable if f(n) is less than or equal to $(1+\varepsilon)\cdot$ max{f(n')}, where n' belongs to the set of nodes which take first places in the OPENED-list.

Another difference between A^* and A_ε lies in the fact that if ,for the node n expanded last, the subset Γ(n) contains no acceptable nodes, then A_ε tries to expand first the nodes from $\Gamma(n)$ then - from $\Gamma(\Gamma(n))$, etc. under an assumption that due to monotonicity of $h(x)$ some nodes will become acceptable together with an increasing $f(n')$.

Algorithm A_ε

```
1. OPENED-list={s}
CLOSED-list=nil   g(s)=0
f(s)=h(s)
ε-threshold=(1+ε)f(s)
Expand (s)
AX:={x ∈ Γ(s) | Acceptable(x)}
/* x is acceptable if x ∈ OPENED-list and f(x) ≤ ε-threshold
2. if AX≠nil then
n=Select AX
else
n=Select OPENED-list
fi
3. Expand (n).
4. If Γ(n) does not contain any acceptable nodes then find Γ(Γ(n)), Γ(Γ(Γ(n))), etc. until either an acceptable node t is found, the OPENED-list becomes empty, or the number N of sequential expanding operations $\frac{\Gamma\ldots\Gamma}{N}$ exceeds some critical value.
Expand(t).
5. AX= {x ∈ Γ(n) | x is acceptable}.
6. If the goal-node is reached then stop the algorithm. If the OPENED-list is nil then there is a general failure else compute new ε-threshold and repeat the steps beginning from step 2.
```

Procedure "select AX" selects the node x with minimal value of f(x) from the nodes belonging to the set AX.

Procedure select_OPENED-list is more complex as it must find an acceptable node n in the OPENED-list such that this node n belongs to the path connecting the last expanded node to the goal-node.

Procedure select_OPENED-list should minimize the criterion

$$\alpha_1 \cdot f(x) + \alpha_2 \cdot h(x)$$

where α_1 and α_2 are selected from the following considerations. The minimization of h(x) permits the goal-node to be reached more quickly. However, the probability of backtracking to the upper levels of search tree also increases. The minimization of f(x) leads to increasing the ε-threshold to retain the current path for further expansion. It increases the number of nodes to be expanded subsequently. The authors of A_ε-algorithm recommend that values of α_2 and α_1 are chosen which satisfy the inequality $\alpha_2 \geq \alpha_1$.

The following results have been obtained for the **traveling** salesman problem with N = 9 cities (see table below).

Table 3.1

ε	**0**	**0.01**	**0.05**	**0.1**	**0.15**	**0.25**	∞
resulting-path cost	100	100.1	100.4	101.1	101.9	103.0	107.0
the number of nodes expanded	100	92	77	54	42	23	15
the number of backtracks	100	33	48	21	13	3	0

From this table one can see that if ε = 0 then the resulting path has minimal cost (100) but the maximal number of the nodes expanded. Together with the augmentation of ε , the number of nodes expanded decreases. For ε=0,25 only 23 nodes have been expanded and only 3 backtracks have been performed. This is significantly less than in the case ε = 0. Note that the loss of accuracy is not higher than 3% for this case.

Another approach to increase the productivity of the heuristic-based searching procces is connected with the use of a probabilistic estimation of the hypothesis that the current path is an optimal one. Consider the approach suggested in [45]. Let T_i denote a tree with the root-node i. Let node n belong to T_i . Suppose that the length (cost) of an optimal path from the root-node of the initial graph T to the goal-node is N. If T_i does not contain an optimal path, then for the node *n* we have

$$\hat{h}(n) = (N - i) + (n - i),$$
$$\hat{g}(n) = n,$$
$$\hat{f}(n) = N + 2n - 2i.$$

Take the function

$$a(n) = \frac{\hat{f}(n) - N}{2n}.$$

One can see that if *n* lies on an optimal path then

$$a(n) = \frac{N - N}{2n} = 0 \text{ as } \hat{f}(n) = N.$$

We can now write that

$$a(n) = 1 - \frac{i}{n}.$$

It is easy to see that $a(n)$ is a random function distributed over the range [0,1]. The idea of the approach [45] is of obtaining samples for given nodes $n_1, n_2, \ldots, n_t$ and testing the hypotheses which regard the means of $a(n_1), a(n_2), \ldots, a(n_t)$ respectively.

3.4.3. Mixed strategies in problem solving

Consider a combination of "branches-and-boundaries" and cutting strategies. For this purpose let us introduce the heuristic function

$$F(x) = WR(x) + FW(x) - LS(x) - FLS(x)$$

where

WR(x) - function evaluating gains obtained at the node x of a search tree;

FW(x) - function estimating maximal future gains expected from this node;

LS(x) - function evaluating losses in the node x; and

FLS(x) - function estimating minimal future losses expected from this node.

F(x) may be represented also in the form

$$F(x) = RG(x) + EG(x)$$

where

$$RG(x) = WR(x) - LS(x),$$
$$EG(x) = FW(x) - FLS(x).$$

Usage of F(x) is based on the following mixed strategy:

(R1) Node y ∈ OPENED-list is chosen if F(y) is maximal among the nodes in that list;

(R2) all the nodes z with the value of F(z) which is less than some value RG(q) of the end-node q (i.e., the node for which either $\Gamma(q)\equiv\varnothing$, or every node t ∈ $\Gamma(q)$ is already contained either in the OPENED-list, or in the CLOSED-list) should be cut down (are not to be expanded). Consider an example below.

Let us maximize the following pseudoboolean function

$$G = 4x1\overline{x2} + 3\overline{x1} + 5x3 + 3\overline{x2x3} + 4x1x3$$

where variables $x_i \in \{0,1\}$.

Start to build a search tree by expanding the initial node S_0 by making two different suppositions: (i) x_1 ∈ solution and (ii) $\overline{x_1}$ ∈ solution.

In case (i) G is reduced to

$$G^{(i)} = 4\overline{x2} + 5x3 + 3\overline{x2x3} + 4x3$$

with LS=3, WR=0

and FW=16, FLS=0

for which F=13.

In case (ii) G is reduced to

$$G^{(ii)} = 5x_3 + 3\overline{x_2}\,\overline{x_3},$$

with LS=8,WR=3,

FW=8, FLS=0

and F=3.

So, proceed from $G^{(i)}$ by making, for example, suppositions for case (iii): x_2 ∈ solution and case (iv): $\overline{x_2}$ ∈ solution.

In case (iii) we have

$$G^{(iii)} = 9x_3$$

and, in respect of the initial function G for a partial solution x_1,x_2 ,

LS=10, WR=0,

FW=9, FLS=0 → F = -1.

By analogy, in case (iv) we have, for the partial solution $x_1\overline{x_2}$,

$$G^{iv} = 9x_3 + 3\overline{x_3}$$

with

LS=3, WR=4,

FW=9, FLS=3

and F=7.

This last case indicates that we are to move on because the corresponding value F^{iv} is maximal among those obtained earlier.

Thus, make the final suppositions:

x_3 ∈ solution (case (v))

$\overline{x_3}$ ∈ solution (case (vi)).

In case (v) we obtain F=7, in case (vi) F=1.

Comparing $F^V=7$ to those obtained earlier one can conclude that this is an optimal solution and all other solutions may be cut off as indicated by rule R2. Hence, an optimal solution is $x1\overline{x2}x3$.

Note that the solving strategy is based on the supposition that

$$\forall x\big(WR(x)+FW(x)+LS(x)+FLS(x)\big)=const$$

and can easily be modified if this condition does not hold.

3.5. Intermediate remarks on heuristics utilization

The heuristic principles together with the corresponding theory of weak methods [16] constitute the basis for heuristic programming. There are some difficulties in the utilization of these principles connected with the problems related to their formalization and an ambiguity of interpretation. Let us consider, for example, a heuristic assertion that "it is better to exclude the objects which probably do not belong to the solution from initial task domain than to include those having good chances to be present in the solution." When solving a minimum-size cover problem on a 0,1-matrix, one may interpret this principle as a step-by-step deleting from the matrix of rows containing the maximum number of zeroes. In the example shown in Fig. 3.15. this gives, after deleting the rows $\alpha_4,\alpha_1,\alpha_6$, the minimum-size covering set $\{\alpha_3,\alpha_2,\alpha_5\}$, with all deterministically included rows. On the contrary, including rows with the minimum number of zeroes may lead to a non-optimal solution: for example, $\{\alpha_1,\alpha_4,\alpha_3,\alpha_5\}$.

	1	2	3	4	5	6	7	8	9
α1	1		1			1	1	1	
α2				1		1	1		
α3			1		1				1
α4				1					1
α5	1	1						1	
α6					1		1		
α7		1						1	

Fig.3.15

From the example above one may conclude that it is necessary to build a suitable interpretation of these principles in every concrete problem. This interpretation should be represented in a quite formal way if it is to be computerized. This means that one needs a definite kind of formal language to represent such principles as those considered before. Let us sum up these preliminary considerations by the following:

(1) there exists a set of heuristics (meta rules) one occasionly refers to while solving (partly) indefinite problems;

(2) to utilize some heuristic(s) one has to make an appropriate interpretation satisfying the conditions required by this heuristic. Since making an interpretation involves a kind of mental activity which is difficult to computerize it is more a human prerogative than a computer's;

(3) to formalize heuristic principles one needs a kind of formal language which is suitable for analysis by logical means.

3.6. Examples of problem solving principles

In this section we consider some principles of problem solving. We avoid building suitable interpretations and postpone that to later considerations. Every principle will be identified by the prefix PRj. The principle PR0 is that considered above.

PR1. An object which causes the greatest "harm" must be put into conditions which are opposite to those corresponding to the most "useful" object.

PR2. To find a good solution one has to eliminate bad ones.

PR3. The decision with the farthest reaching consequencies must be taken the first.

PR4. The unknown problem may be converted into the known one dually either through a conditions transformation or through a task domain transformation. One needs to find equivalent transformations (i.e., transformations which do not lead to losing the solution). If one fails, then attempts are to be made to find transformations with "minimum discrepancy" between initial and transformed tasks.

PR5. The task may be solved either in by moving from what is known to what is unknown (is required to be found) or in the opposite direction.

PR6. (Every) task is connected with hidden regularity. Define independent and dependent factors. To clarify the nature of regularity try some simple and deterministic examples. Collect, if necessary, some statistical material and analyze it. Try to reduce the number of "independent factors", for example, by replacing factors X_1 and X_2 by the third (synthesized) factor X_3 .

PR7. Similar tasks are solved by similar methods.

PR8. To determine the influence of every independent factor $X_i \in \{X_1, X_2, \ldots, X_n\}$ one has to fix some of the factors and to vary the others.

PR9. If the required solution has no property P, then all variants should be declined which imply P.

PR10. On the contrary, if the required solution has a property P and a particular variant W implies $\overline{P}$ then W should be declined.

PR11. The idea of greedy algorithms: to find an optimum solution at every step is correct provided that it does not lead to loss of a global optimum solution after each step. It means, as in the example of a minimum-size covering set for a given 0,1-matrix, that one has to establish all the necessary conditions for providing the correctness of deleting or including each row in the solution.

PR12. If there is a factor X violating a given regularity then its influence may be restricted in the following ways

12.1) deleting factor (object) X if and only if it does not lead to the loss of a solution.

12.2) replacing X by Y, where Y is a new object performing all the functions of X, and being neutral in respect to the problem solution.

12.3) introducing a "positive" factor Z which suppresses the "negative" factor X or neutralizes its negative influence.

PR13. Improve, first of all, those factors affecting the criterion C which have "the worst" values in respect to the optimum meaning of C.

PR14. In order to provide a restricted "try-and-test"-method one has to know some applicable principle of choosing a solution. The more precise is this principle, the less is the computational complexity of the "try-and-test" method.

PR15. It is necessary to seek such conditions, which after being removed, generate the minimum number of new alternatives, or, if being admitted, provide a reduction in the maximum number of alternatives.

PR16. Let F be a set of conditions given on the task domain, f be the additional conditions, and $\{F \cup f\}$ be the conditions for some particular case of an initial problem with known solution R. The latter is also a solution for a common problem with condition set F if

$$F \vdash f.$$

One may consider R basic solution if the following is true:

$$not(F \vdash f)\ \&\ not(F \vdash \overline{f}).$$

The basic solution may be not an optimal one but it does not contradict problem conditions.

PR17. Simple problems make up the complicated ones, that is, a complicated problem may be or may not be recursively represented by means of more simple problems. However, we cannot assert that a recursive representation is sufficient for the problem being resolvable effectively. To be more precise, consider an example. The sum

$$S_n = 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$$

may be recursively reduced to

$$S_{n^2} = 1^2 + 2^2 + \ldots + n^2$$

and vice versa, i.e:

$$n^2 = 2 \cdot S_n - n$$

$$(n-1)^2 = 2\cdot S_{n-1} -(n-1)$$

$$(n-2)^2 = 2\cdot S_{n-2} -(n-2)$$

$$\ldots$$

$$1^2 = 2\cdot S_1 - 1$$

or

$$S_{n^2} = 2\sum_{i-1}^{n} S_i - S_n = S_n + 2\sum_{i=1}^{n-1} S_i .$$

Taking into account that

$$S_n = S_{n-1} + n$$

we obtain a recursive representation of S_{n^2} through $S_n, S_{n-1}, \ldots S_1$ with the help of a primitive recursion operator and function f(x)=x+1.

PR18. The extreme cases either distort the nature of a regularity or make it more transparent to the investigator. This principle is in accordance with the corresponding Backon's maxima and recommends either to avoid "atypical" cases or to find such forms which reveal as much as possible of the essense of a problem.

PR19. One may think of some factor as

a) of known entity with known nature;

b) of known entity with unknown nature;

c) of unknown entity with known outer effects;

d) of unknown entity with unknown nature.

In case b) one may determine the nature of the factor with the help of experiment(s). In general, a solution may be sought according to the alternatives in principle PR15. A set of alternatives should form a full group of alternatives. **In** case c) one should clear the conditions necessary for the factor being activated, since it may be sufficient to know, at least partly, the nature of the unknown **factor** **which** reflects under given conditions. To realize this idea, divide the initial problem into subproblems. Consider a subproblem which contains an unknown factor. Adhere to the following scheme of deducing the properties of the unknown factor:

$$\begin{tikzcd}
\text{(property X)} \arrow[r] \arrow[dr] & \text{(action X)} \arrow[d] \\
 & \text{(sequence Y)}
\end{tikzcd}$$

Use the inference rules:

$$\frac{X \to Y, \bar{Y}}{\bar{X}}$$

which defines which properties do not belong to factor, and

$$\frac{X \to Y, Y}{(presumably) X}$$

enables one to make an alternative for principle PR15.

Let factor X violate (contradict) regularity Y. Its negative influence may be eliminated by:

a) deleting factor X;

b) neutralizing factor X;

c) introducing a positive factor Z which supresses factor X.

Point b) may be realized (partly or completely) by:

b.1) relaxing an initial criterion;

b.2) replacing an initial criterion by the other;

b.3) separating X from Y (or removing the linkage between these two factors);

b.4) introducing new factor X' which is in antagonism with negative factor X;

b.5) replacing X by the new alternative X' not violating Y;

b.6) defining constituent parts of X with applying neutralization procedure to these parts;

b.7) weakening factor X.

Point a) is related to:

a.1) the definition of the constituent parts of X and removing the most essential of these;

a.2) a move to the system where X is absent and Y is present;

a.3) reducing X to the state where it loses the negative properties;

a.4) introducing a new factor X' maintaining system integrity after the removal of factor X;

a.5) move to the antisystem, where $\overline{X}$ and $\overline{Y}$ are mutually concerted and developing $\overline{X}$.

Point c) may be realized by

c.1) move to a wider system;

c.2) finding a factor which motivates X;

c.3) finding a factor which motivates Y.

Our following task is to illustrate some of the above listed principles with examples of the corresponding problems. Note, that we are not pretending to consider as many principles as possible but we wish to demonstrate that these principles really form the basis of the solving procedure, i.e. constitute the concept of the approach to finding a solution.

3.7. Solution tree

Consider the 5-tuple:

$$<S_I, S_F, A, R_A, R_S> \qquad \textbf{(3.0.)}$$

where S_I - is an initial system state;

S_F is a final system state;

A is an algorithm providing the mapping $S_I \rightarrow S_F$;

R_A is a list of restrictions on algorithm A realization;

R_S is a criterion (list of restrictions) which must be fulfilled in S_F .

The state S_J , directly reached from state S_K , is called the nearest successor of S_K, and the state S_K is the nearest predecessor of S_J . Graph $\Gamma(S,\Pi)$ with the set S of system states $S_J \in S$ and the set Π of edges connecting all the vertices with their nearest successors, forms a solution tree with root S_I and leaf S_F , if

1)the μ-th (μ>0) level in Γ is formed by the nearest successors of (μ-1)-th level which do not belong to the levels (μ-2),(μ-3),...,(0);

2) the leaves in Γ correspond to those states in which all the nearest successors are contained in the upper levels in Γ (the exception, maybe, is for the leaf S_F). The possible problem formulations for (**3.0**) are the following :

a) for given S_I, R_A, R_S , find the corresponding S_F and A ;

b) for given S_I, S_F, R_A, find A ;

c) for given S_I, R_S, find S_F.

Let us focus our attention on the formulation a). In this case the combinatorial character of the problem is expressed the most clearly; the crux of the matter is in seeking an optimal way in Γ from the node S_I to (some) leaf S_F satisfying R_S The total number Z of all possible ways from S_I passing through levels 1,...,m is

$$|Z| = |Z_1| \cdot |Z_2| \cdot \ldots \cdot |Z_m|$$

where $|Z_i|$ - is the cardinality of the set of nodes in the i-th level.

Since an unrestricted " try-and-test " method for large Z is practically ineffective, we start our considerations with the *restricted* and *directed* "try-and-test " - principle.

3.7.1. Restricted and directed "try and test" - principle

The restricted and directed "try-and-test" - principle enables us to smooth out some of the drawbacks of the original algorithm, e.g., those connected with the choice of a single general criterion CR for the multicriterium optimization task. From the formal viewpoint, the solution is searched for on the partial solution graph $\Gamma' \subset \Gamma$ with every level containing alternative vertices , i.e. vertices which admit a set of alternatives in the sense of compared values of criterion CR. Let CR represent a function of arguments $\alpha_1, \alpha_2, \ldots, \alpha_t$.

State-vertex S_K dominates over state-vertex $S_1 (S_K \rhd S_1)$, if S_K and S_1 belong to the same level in Г' and all arguments (parameters) $\alpha_1^k, \alpha_2^k, \ldots, \alpha_t^k$ characterizing S_k are equal or better than corresponding parameters $\alpha_1^l, \alpha_2^l, \ldots, \alpha_t^l$ characterizing state-vertex S_l. In this case, a chosen set consists of such vertices S_x, S_y , that $S_x \not\rhd S_y$ and $S_y \not\rhd S_x$ and there is no vertex S_k which does not belongs to the chosen set and such that $S_k \rhd S_x$ or (and) $S_k \rhd S_y$. On the basis of the restricted "try-and-test" - principle ,one may improve (in the sense of R_s from **(3.0)**) the solution of a problem "a") obtained with some known heuristic algorithm A'. The scheme of the corresponding procedure in this case may be as follows :

S.0. There are given S_1, R_A, R_S, A'. Set $\mu = 0,\ S_\mu = S_I$.

S.1. If S_μ is a leaf, then stop: $S_F = S_\mu$, else go to step 2.

S.2. In the level $(\mu + 1)$ determine a chosen set $C_{\mu+1}$. Let $S_j \in C_{\mu+1}$.

Suppose, that in the 5-tupple **(3.0)**, $S_I = S_j$ and find S_F with the help of algorithm A'. Repeat step S.2. for all other state-vertices from $C_{\mu+1}$.

S.3. Go from state-vertex S_μ of the μ-th level to the vertex S_t in the level $(\mu + 1)$ for which a final state S_F, found by algorithm A', is better (in the sense of R_S) than any other final state S_F , reached from the state S_t, belonging to $(\mu + 1)$-th level.

Set $\mu = \mu + 1,\ S_\mu = S_t$, go to step 1.

Taking into account that the set of all routes obtained by this procedure contains the route found by algorithm A' the final solution cannot be worse than that one.

Example. Consider a dead-lock problem in the system of parallel abstract processes in the following interpretation.

Let there be given abstract processes $P_1, P_2, \ldots$. Every process P_i is characterized by the corresponding realization U_i:

$$U_i = <\succ, R_i^j>,$$
$$j = 1, 2, \ldots, \omega_i$$

where $\succ$ - is a precedence relation ;

$R_i^j \subseteq R$ - is the (sub)set of resources process P_i needs at step j and $R_i^j \succ R_i^{j+1}$ denotes the separation of two sequential subsets of resources in realization U_i.

Model $<\succ, R_i^j>$ defines a system of abstract parallel processes as follows :

E1. Every process P_i subsequently realizes steps j (j=1,2,...) of the corresponding realization U_i (i=1,2, ...) starting from step j=1.

E2. A move to next step (j + 1) from step j is possible if every resource in $R^{(j+1)}$ possesses sufficient capacity (number of resource units). (We shall suppose that every process requires one resource unit of every resource in the corresponding subset R^j).

E3. At any one time the process may realize one and only one step of the corresponding realization. It is also insisted that a move from one step to the next step is performed for only one process at a time.

In the senses of E1-E3 a dead-lock corresponds to the case when there are no processes satisfying E2 and have yet not fulfilled their realizations.

The state S_t of the model $U\_ = <\succ, R^j\_>$ corresponds to the set of pairs

{ <x,N(x)>,...,<y,N(y)> },

where x(y) - is an indentifier of process Px (Py) ;

N(x) (N(y)) - is the number of the step being realized by process Px (Py) at moment t.

A state S_{t+1} is the nearest successor of a state S_t if it differs from S_t by only the pair <i.N(i)> and S_{t+1} is obtained from S_t according to the rules E1-E3. In this case we call states S_t and S_{t+1} adjacent and denote this fact by $S_t \succ S_{t+1}$.

Definition. Any deterministic rule A which generates for a given state S_t an adjacent state S_{t+1} not violating E1-E3 and not introducing a new process into the system is called a **V**-rule. The simplest examples of **V**-rules are FIFO [46], LIFO ,etc.

Using a **V**-rule and the scheme of a restricted and directed "try-and-test" method the problem of preventing a dead-lock may be solved as follows. Let the current state of model $<\succ, R_i^j>$ be S_t. Then moving from S_t to adjacent state $S_{t+1}(S_t \succ S_{t+1})$ is allowed if in chain

$$S_{t+1} \overset{V}{\succ} S_{t+2} \overset{V}{\succ} \dots \overset{V}{\succ} S_{t+m}$$

the final state S_{t+m} reached from S_t after m steps with the help of the V-rule is empty, i.e. $S_{t+m}=\{\varnothing\}$. It simply means that in S_{t+m} all processes have finished their realizations. Finding the best (in some sense) V-rule is essential for the synchronization strategies used in a local network protocols and process monitoring [47]. Note, that one of the main requirements to these algorithms is providing the minimum total time for realizing all processes.

3.7.2. Cutting the worst variants

This principle has the most effective utilization in the "branches-and-bounds" - method [48]. Intuitively, its idea is in reduction of search space by deleting the purposeless and the worst, from the viewpoint of a heuristic evaluation function, state-vertices in the solution graph. Rather often this principle may be complementary to the previous one. Let us consider some illustrative examples of this principle. Consider the minimum-cost covering tree problem, which involves a weighted graph and the search for a subgraph that spans the graph and has minimal cost. The resulting tree must contain the edges forming a minimum-cost cover of the graph nodes. To apply the cutting principle, one needs to know the property characterizing the solution. Suppose some minimum-cost covering tree is that shown in Fig. 3.16.

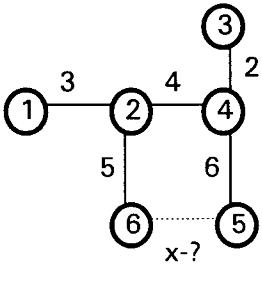


Fig. 3.16.

Let us assume that nodes 5,6 are connected by an edge in the initial graph. What can one say about the length of the edge (5,6)? May the length of a dotted line connecting nodes 5 and 6 be less than 6 ? The answer to the question can only be negative. Modify Fig. 3.16. In the new Fig. 3.17. length X cannot be less than 5. These relations are established according to the following speculations.

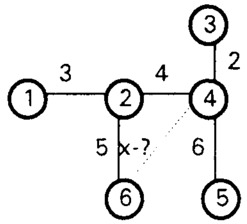


Fig. 3.17.

Consider again Fig. 3.16. On the supposition that the dotted edge exists, its length X must satisfy the inequality

$$X \geq \max_{\varphi}\{U_{\varphi}\},$$

where U_{φ} - is a length (weight) of the edge φ such that it belongs to the same cycle as a dotted edge (4,5) or (5,6).This is exactly the property we require. It enables us to delete from the graph the edges with the largest weights if they belong to any cycle. Proof may be obtained as follows.

For every coherent graph with n=1 cycles, the assertion is evident. Suppose it is correct for any graph with $n \geq 2$ simple cycles. The cycle is simple if it does not contain other cycle(s). Connect any two nodes to get (n+1) simple cycles. Let an additional edge be α_x . If this does not belong to the resulting covering tree then the assertion is correct due to supposition for $n \geq 2$ simple cycle-graphs. Suppose α_x belongs to a minimum-cost covering tree (i.e. the tree with the same nodes as in initial graph). Then in the last remaining cycle containing α_x $U_{\alpha x}$ cannot be the largest among the nodes forming this singe cycle. It simply means that the weight U_{ax} of the edge α_x may vary within the bounds $\left(0, U_{\overline{\max}}\right]$, where

$$U_{\overline{\max}}=\min\{\max_{j\in J} U_j\}$$

and max U_j is the maximal edge weight in the j-th cycle containing α_x . This property of α_x may be extended to other edges except that having the maximum weight and belonging to any cycle since the supposition that this edge belongs to a resulting tree leads to a contradiction. So, we obtain a cutting principle for the minimum-cost covering tree problem, i.e. "delete subsequently the edges with the maximum weight if such edges belong to any cycle in the graph".

The second example of this kind may be that concerning the "Traveling Salesman Problem" which was investigated earlier.

3.7.3. Principle of suitability

This and the following principles deal directly with the precedence relation ($\succ$) on the set O of e.s.o. The whole problem is formulated as finding optimal permutation on O. There are two possible tasks connected with the relation $\succ$. The first one requires us to impose an optimal precedence relation on the given set O of e.s.o ,to provide the given result. In other words, it requires us to generate a precedence relation in some optimal way. The second task requires us to order a partly ordered set of e.s.o to ensure some given criterion. This second variant may be considered as a generalization of the first one as it deals with a partly ordered set of e.s.o such that for any pair of e.s.o X and Y neither $X\succ Y$ nor $Y\succ X$ may hold.

An interpretation of the suitability principle was previously formulated as the statement that "an object which causes the greatest harm must be put into those conditions which are opposite to those corresponding to the most "useful" object. Denote this principle as the **C**-principle. Suppose there are N positions and N objects, and it is known, for every object i, a quantitative estimation $f_i(.)$ of the harm caused by this object and for every position Π(i) there is known the "degree of satisfaction" g(Π (i)) of negative influence connected with assignment object i **to** the position Π(i). Hence,

$$\min \sum_{i=1}^{N} f_i \cdot g(\Pi(i))$$

corresponds to such an assignment of objects 1...N to positions П(1)...П(N) that maximal estimations $f_i(.)$ correspond to minimal values of g(П(i)).

Conversely, if the effect of assigning objects to the positions is evaluated by the "positive" influence $f_i(.)$ made by the objects i ($i=\overline{1,N}$) (i.e., the usefulness of object's assignment to positions is estimated), then

$$\max \sum_{i=1}^{n} f_i \cdot g(\Pi(i))$$

is guaranteed by such an assignment when maximal $f_i(.)$ correspond to maximal values of g(П(i)). Consider some examples. Let there be given an unordered set A of independent jobs: A=$\{a_1,\dots,a_n\}$ with the corresponding working times $t_1,\dots,t_n$ and due dates $D_1,\dots,D_n$. Every job a_i is completed at time τ_i and it is required to find an optimal linear sequence of operations to ensure

$$F = \max_i[\max(0;\tau_i - D_i)] \to \min, \qquad (3.1)$$

We may interpret the set A as programming modules and the initial problem as the known "N jobs- one computer" scheduling problem with given due dates and criterion (3.1).The principle of suitability in our case may be realized as follows:

(a) One determines the most "negative" (in the sense of (3.1)) object among those not assigned to the positions.

(b) The object selected at the previous step is assigned to the free position which neutralizes its negative influence as much as possible.

From the view of point (b),the schedule we are intrested in has the property that its final (the most righthand) position neutralizes any negative influence caused by the assignment of the object to it. It means that the object which occupies the rightmost position in the resulting schedule has no influence on the other objects. Therefore, this position may be used in the sense of the point (b) above. It remains to locate such a job which corresponds to this rightmost position.

Obviously, the value of $\tau_{a(N)}$ of the job $a(N)$ which occupies the rightmost position N does not depend on the length of the schedule at all as it is equal to the sum $\sum_{i=1}^{N} t_i = const$. Thus, (3.1) requires

$$\max\left[0;\left(\sum_{i=1}^{N} t_i - D_{a(N)}\right)\right] \to \min.$$

It follows automatically that $D_{a(N)}$ should be the maximal among $D_1,\dots,D_n$ (more accurately, the value of $D_{a(N)}$ should satisfy the relation

$$D_{a(N)} \geq \sum_{i=1}^{n} t_i$$

but the former relationship assumes the latter one).

Example 2. Let X and Y be two sequential service devices and A be the set of jobs: $A_1 = \{a_1,\dots,a_n\}$. Every job from A is firstly executed on device X and then is transferred to device Y. There are known times t_{ix} and t_{iy} required by every job a_i to be executed correspondingly on devices X and Y. The problem becomes that of finding the final schedule P satisfying the criterion

$$F' = \min \max_{i}\{\tau_i\},$$

where τ_i - is a completion time of the job a_i.

Define how each job assigned to the next free position in Π makes an influence on F'. Suppose that device X becomes free at moment Λ_x when some job a_ε leaves it. Also suppose that this job leaves the device Y at moment $\Lambda_y > \Lambda_x$. Precisely at moment Λ_x device X is occupied by the next job a_γ which releases it at moment

$$\Lambda'_x = \Lambda_x + t_{\gamma x}.$$

One may consider the assignment of job a_γ after job a_ε to be bad if $\Lambda'_x > \Lambda_y$, as in the interval $[\Lambda_x, \Lambda_y]$ the device Y would be unoccupied by any job. In accordance with the principle under consideration one should neutralize the negative influence of job a_γ by demanding that $\Lambda'_x \leq \Lambda_y$. Having in mind the form of F' it should also be required that

$$\max\left[\left(\Lambda_x + t_{\gamma x}\right); \Lambda_y\right] + t_{\gamma y} \to \min .$$

It, further, gives

$$\Lambda_x \to \min .$$

Obviously, the condition obtained must be kept at all times within the interval occupied by Π. The condition

$$\Lambda'_x \leq \Lambda_y$$

may be preserved by the strategy illustrated in Fig. 3.18, and the condition

$$\Lambda'_x \to \min$$

may be preserved by the corresponding strategy shown in Fig. 3.19. In fact, this is all that is required by an algorithm as suggested in [18] .

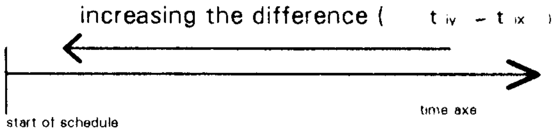


Fig. 3.18.

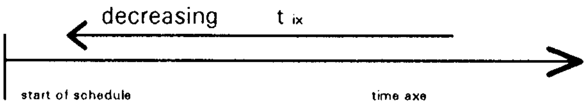


Fig. 3.19.

3.7.4. Principle of minimization of aftereffect

To illustrate this principle consider the multiprocessor scheduling problem in the following formulation. Let there be given the system $<P,\succ>$, where $P=\{p_i\}$, $i=\overline{1,m}$ - is the set of programming modules and $\succ$ - is a partial order relation imposing a precedence on P, i.e. $p_i \succ p_j$ if module p_j cannot start executing before p_i has finished. There are also given the execution times of modules from P on the corresponding processors 1,2,...,n.

We are interested in the schedule realizing an acceptable module assignment to the processors and providing

$$\min \max_{i=1...n}\{\tau_i\} \qquad \textbf{(3.2)}$$

where τ_i - is the time at which processor i completes execution of the last module assigned to it. The value of τ_i also contains the lengths of the idle intervals (if any) when processor i does not perform any work because of the restrictions imposed by the precedence relation . First of all, the principle we are exploring requires us to clarify the nature of the influence on the final criterion (**3.2**) of each module assignment to the corresponding computer. To do this, one should destinguish between two aspects connected with this principle:

(i) in the first attempt the modules are assigned to processors which cause the greatest influence on the assignment of other modules (this aspect forms the basis for known scheduling strategies [48]);

(ii) the estimation is necessary in order to evaluate the effect of module assignment on the resulting criterion (**3.2**).

From the viewpoint of the second aspect, the idea of known strategies to choose the critical path modules in graph coding the pair $(P,\succ)$ in the first attempt is not the best. In [50] an algorithm is proposed which takes into account for each module p_i the corresponding sum

$$T(p_i) = \sum_{j \in Suc(p_i)} t_j$$

where, t_j - execution time of module j ;

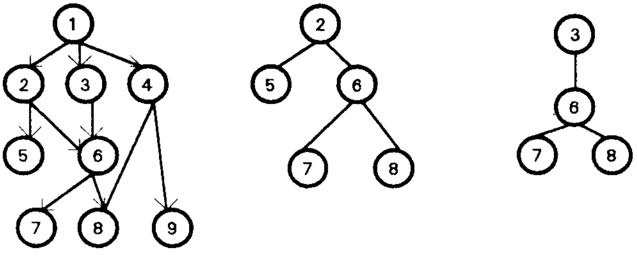
Suc(p_i) - the set of modules connected to module p_i by the paths starting from p_i in the precedence graph coding <P,$\succ$>.

This approach, however, does not discriminate between two different vertices p_a and p_b in the precedence graph with equal values $T(p_a) = T(p_b)$ the difference between the topologies of subgraphs

$$\left\langle \left\{ p_a \bigcup Suc(p_a) \right\}, \succ \right\rangle and \left\langle \left\{ p_b \bigcup Suc(p_b) \right\}, \succ \right\rangle.$$

Let us show how to use the aspect (i) above to find the necessary estimations for modules.

Denote the precedence graph,coding <P,$\succ$> ,by G(**P**,$\overline{U}$) with the vertices P and arcs $\overline{U}$ such that$\overrightarrow{(p_i, p_j)} \in \overline{U}$ if $p_i \succ p_j$ .Call the subgraph $< \{p_x \cup Suc(p_x)\}, \succ>$ p_x-subgraph (see examples in Fig. 3.20).



Graph G(P,U) P2-subgraph P3-subgraph

Fig.3.20

Let L_0 be the schedule length, i.e.

$$L_0 = \max_{i=1...n} \{\tau_i\}$$

and L_x be the length of the schedule built for the p_x - subgraph. Consider a homogeneous multiprocessor system (for a heterogeneous multiprocessor system ,all the results obtained here may be easily reproduced by analogy). Then, the strategy of module assignment based on the estimations of L_j and providing minimum L_0 satisfies the conditions below:

(i) to assign module j to a processor, all predecessors in G should be executed;

(ii) the corresponding value L_j computed for P_j - subgraph should be maximal among other modules satisfying p. (i);

(iii) each module is assigned to the idle processor which completes its execution before the others.

One may directly use this scheme to find all the estimations L_j. Indeed, in order to find L_k one has to build the schedules for p_l -subgraphs $(l \neq k)$ such that $p_l \subseteq p_k$ (i. e. p_l is a subgraph of p_k-subgraph). So, one has a recursive scheme where the estimations of L_r for the vertices having no successors are equal to the corresponding execution times t_r . Instead of formal algorithms, consider the example in Fig. 3.21 for the homogeneous 2-processors system (the numbers near the vertices define execution times).

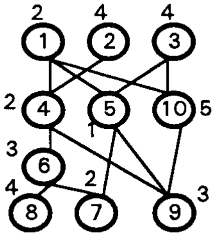


Fig. 3.21.

First, one establishes

$$L_9 = 3, L_8 = 4, L_7 = 2$$

Now it is possible to find L_6, L_5 and L_{10}.That is, having built the corresponding schedules for p_6, p_5 and p_{10}- subgraphs,it is possible to obtain:

$L_6 = 7, L_5 = 4, L_{10} = 8.$

Further, it may be deduced that $L_4 = 9$ and $L_3 = 12$.

As an illustration, the schedule for the p_4-subgraph obtained on the basis of estimations L_6, L_7, L_8 and L_9 is shown in Fig. 3.22.

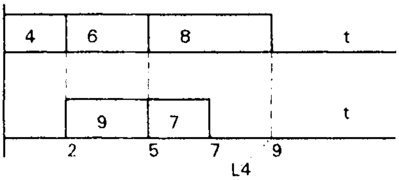


Fig. 3.22.

The resulting estimations L_i of all modules are shown in Fig. 3.23.

modules	1	2	3	4	5	6	7	8	9	10
bj	11	13	12	9	4	7	2	4	3	8
Lj	13	13	12	9	4	7	2	4	3	8

Fig. 3.23a

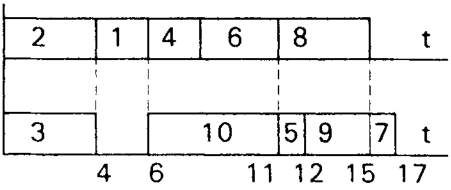


Fig. 3.23b): b_j - based schedule

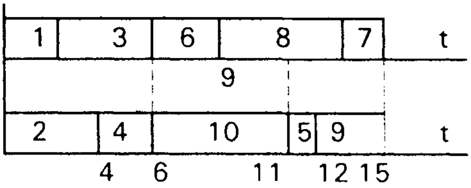


Fig. 3.23c): L_j - based schedule

Also given are the estimations obtained in accordance with the formula [49] :

$$\begin{cases} b_j = t_j + \max\limits_{pk \in Suc(p_j)} \{b_k\} \\ b_j = t_j \;\; if \; Suc(p_j) = \varnothing . \end{cases}$$

The only (!) difference between L_1 and b_1 provides the better final result in the sense of (**3.2**).

The principle of the minimization of an aftereffect on the basis of the estimations L_j has proved to be more effective than that based on the estimations b_j.The following results were obtained for 27 arbitrary graphs and 2 processors : in 3 cases the resulting length L_0 was shorter for homogeneous systems and in 2 cases - for heterogeneous. The relative retraction of the length L_0 was maximal for heterogeneous systems and as much as 23,6 % of the length L_0 obtained on the b_j-estimations basis. There were no cases when making use of b_j estimations resulted in more reduced schedule length. These results show that the efficiency of the principle of the minimization of an aftereffect depends on evaluating the consequencies of module assignment to processors.

3.7.5. The "Maxmin"-principle

The essence of the "maxmin" - principle is intuitively understood as improving those factors affecting the integrated criterion C which has "the worst" values in respect to optimal meaning of C. To clarify this principle, let us consider an explanatory example known as the containers packing problem [51]. Suppose, there are n resources (containers) $r_1,r_2,...,r_n$ each with capacity $\rho = 1$. There are also processes i = 1,2,...,N. Every process i requires a definite part of each resource j: $0 \le \rho_{ij} \le 1$. Further ρ_{ij} will be referred to as the requirement of process i for a given part of resource j (in conditional units). Introduce the requirements matrix $\rho = \left[\rho_{ij}\right]$. Let the problem consist in finding a maximum number $N^{\max}$ of parallel processes which can be executed on resources $r_1, r_2, \ldots, r_n$ provided that

$$\overset{\vee}{f}_j = \sum_{i \in N^{\max}} \rho_{ij} \le 1 \text{ for all j=1,2,...,n}$$

With the help of the "maxmin" - principle one can find a heuristic solution as follows.

Step 1. If there is at least one column j in the requirements matrix ρ which violates the constraints $\overset{\vee}{f}_j$ then go to step 2; otherwise stop: the solution consists of the number of rows remaining undeleted in ρ .

Step 2. Let there remain rows $I = \{i_1, i_2, \dots, i_k\}$ in ρ . Delete the row $i^* \in I$ providing

$$\max_{j=1\dots n} \overset{\vee}{f}_j \to \min .$$

Go to step 2.

Example. Let requirements matrix ρ be given in the following form $\rho =$

processes	**resources**		
	r1	**r2**	**r3**
1	0.2	0	0.7
2	0.5	0.4	0.3
3	0.2	0	0
4	0.5	0.3	0.4
5	0	0.5	0.1
6	0.3	0.2	0.2
Σ	1.7	1.4	1.7

Deleting row 1 one obtains the following vector of elements sums in each column: $\sum_1 = (1.5; 1.4; 1.0)$ and maximal sum $V^1_{\max}$ equal to 1.5 . By analogy for every row 2,3,...,6, the following vectors may be obtained:

$$\sum\nolimits_2 = (1.2;1.0;1.4), V_{\max}^2 = 1.4$$
$$\sum\nolimits_3 = (1.5;1.4;1.7), V_{\max}^3 = 1.7$$
$$\sum\nolimits_4 = (1.2;1.1;1.3), V_{\max}^4 = 1.3$$
$$\sum\nolimits_5 = (1.7;0.9;1.6), V_{\max}^5 = 1.7$$
$$\sum\nolimits_6 = (1.4;1.2;1.5), V_{\max}^6 = 1.5.$$

It is necessary to select a row which provides the minimal value of $V_{\max}^i$ among rows j=1,2,...,6. Thus, the first selected row is row 4 as this provides $V_{\max}^4 = 1.3$ which is a minimum one. Row 4 is deleted from the requirements matrix. Repeating the procedure by analogy, one obtains the final solution in the following form $N^{\max}$:

	resources		
processes	**r1**	**r2**	**r3**
1	0.2	0	0.7
3	0.2	0	0
5	0	0.5	0.1
6	0.3	0.2	0.2
Σ	0.7	0.4	0

Example 2. Let the set $I = \{i_1, i_2, \ldots, i_n\}$ of indepent tasks be given. Let there be $k \geq 2$ different (modules) processors. The matrix $\left[t_{ij}\right]$ $i = \overline{1,n}, j = \overline{1,k}$ of operating times of tasks i on computers j is known. The problem consists of making an optimal schedule for the execution of tasks employing criterion (**3.2**).

The "Maxmin" - principle enables one to realize the following solution scheme [52]:

- at every step of the planning procedure, module i_s is selected from the modules not yet scheduled, which satisfies the following conditions:

(i) the value

$$\overset{\vee}{f} = \tau_k^* + t_{sk}$$

where τ_k^* is a current working time of processor k is the minimum among $\tau_1, \tau_2, \ldots, \tau_k$ and

(ii) this value is the maximum among all unscheduled modules S.

More precisely,

$$\check{f} = \max_{s} \min_{k} (\tau_k^* + t_{sk})$$

Let $[t_{ij}]$:

processors			
modules	p1	p2	p3
1	5	4	6
2	3	8	2
3	4	3	5
4	2	4	4

First set $\tau_1^* = \tau_2^* = \tau_3^* = 0$.

We have:

$$S = 1 \ \check{f}_1 = \quad \min_k \left(\tau_1^* + t_{11} ; \tau_2^* + t_{12} ; \tau_3^* + t_{13} \right) = 4$$

$$S = 2 \ \check{f}_2 = \quad \min_k \left(\tau_1^* + t_{21} ; \tau_2^* + t_{22} ; \tau_3^* + t_{23} \right) = 2$$

$$S = 3 \ \check{f}_3 = \quad \min_k \left(\tau_1^* + t_{31} ; \tau_2^* + t_{32} ; \tau_3^* + t_{33} \right) = 3$$

$$S = 4 \ \check{f}_4 = \quad \min_k \left(\tau_1^* + t_{41} ; \tau_2^* + t_{42} ; \tau_3^* + t_{43} \right) = 2$$

$$\max_S \check{f}_S = 4.$$

The above result corresponds to the assignment of module 1 to processor p_2 :

$$\tau_2^* = \tau_2^* + t_{12} = 4.$$

Delete the row 1 from matrix $\left[t_{ij}\right]$ and repeat the procedure by analogy.

$$S = 2\ \overset{\vee}{f}_2 = \quad \min_k\left(\tau_1^* + t_{21}; \tau_2^* + t_{22}; \tau_3^* + t_{23}\right) = 2$$

$$S = 3\ \overset{\vee}{f}_3 = \quad \min_k\left(\tau_1^* + t_{31}; \tau_2^* + t_{32}; \tau_3^* + t_{33}\right) = 4$$

$$S = 4\ \overset{\vee}{f}_4 = \quad \min_k\left(\tau_1^* + t_{41}; \tau_2^* + t_{42}; \tau_3^* + t_{43}\right) = 2$$

$$\max_S\left(\cdot \overset{\vee}{f}_2, \overset{\vee}{f}_3, \overset{\vee}{f}_4\right) = 4,$$

which corresponds to the assignment of module 3 to processor p_1. Set

$$\tau_1^* = \tau_1^* + t_{31} = 4.$$

Further, we obtain

$$S = 2\ \overset{\vee}{f}_2 = 2$$

$$S = 4\ \overset{\vee}{f}_4 = 4$$

Assign module 4 to p_3:

$$\tau_3^* = \tau_3^* + t_{43} = 4$$

and finally S=2 $\overset{\vee}{f}_2 = 6$: assign module 2 to processor p_3. The resulting schedule has the form shown in Fig. 3.24.

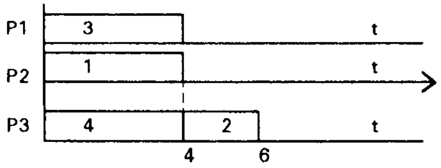


Fig. 3.24.

3.7.6. "Greedy" algorithm's schemata

The basic idea of the greedy algorithms is rather simple: to choose the best continuation from every problem state. This strategy is successful provided it keeps the optimal solution within the bounds of possibility. To clarify this idea let us consider the shortest path problem given on the graph. Consider the following relations:

$X_2 = X_1 + 4$ $\qquad X_6 = X_2 + 2 \qquad X_9 = X_7 + 4$

$X_3 = X_1 + 3$ $\qquad X_7 = X_3 + 2 \qquad X_{10} = X_7 + 5$

$X_4 = X_1 + 5$ $\qquad X_8 = X_4 + 3 \qquad X_{11} = X_5 + 1$

$X_5 = X_1 + 6$

$X_{12} = X_9 + 6 = X_{10} + 4 = X_{11} + 2$

where X_i stands for the minimum-length path connecting vertices x_1 and x_i in a graph. We use the following common relationship:

$$S^*(i,k) = \min_j (S_{i,j} + d_{jk}),$$

where $S^*(i,k)$ is the minimum total length path, connecting vertices i and k;

d_{jk} is the length of the arc (j,k).

This relation may be applied directly to the final step of the algorithm. Now we can write

$$X_{12} = \min(X_9 + 6; X_{10} + 4; X_{11} + 2) \qquad \text{(i)}$$

$$X_9 = \min(X_7 + 4) \qquad \text{(ii)}$$

$$X_{10} = \min(X_7 + 5; X_8 + 2) \qquad \text{(iii)}$$

$$X_{11} = \min(X_5 + 1) \qquad \text{(iv)}$$

$$X_7 = \min(X_3 + 2) \qquad \text{(v)}$$

$$X_8 = \min(X_4 + 3) \qquad \text{(vi)}$$

$$X_5 = \min(X_1 + 6) \qquad \text{(vii)}$$

$$X_3 = \min(X_1 + 3). \qquad \text{(ix)}$$

Processing these relations in reverse order, one can obtain $X_1 = 0; X_4 = 5; X_3 = 3; X_5 = 6; X_8 = 8; X_7 = 5; X_{11} = 7; X_{10} = 10; X_9 = 9; X_{12} = 9$. Thus, the minimum length path we are interested in has total length = 9. It is not difficult to find the path itself. For example, considering the relation

$$X_{12} = \min(X_9 + 6; X_{10} + 4; X_{11} + 2)$$

where X_9, X_{10} and X_{11} are already known, one simply finds that the last but one vertex in the searched path is 11, etc.

The correctness of the "greedy" algorithm principle may be provided by additional considerations for the last step performed by the algorithm. This is valid due to the fact that finding the optimal solution at the last step ensures a global optimum provided that the previous path is an optimal one. The last relation lies in the basis of dynamic programming optimum principle [53]. Denote the control chosen at the step i by X_i . Then one may write

$$L_i^* = \underset{X_i \in \psi_i}{extr} (L_{i-1} + f(X_i)),$$

where $L_i = \sum_{j=1}^{i-1} f(X_j)$ and ψ_i is a definition space of X_i

Bellman's optimum principle says that the value of X_i should be optimal independently of the way it was reached. Let us consider the next example, which requires one to find the values of unknowns X_1, X_2, X_3 providing

$$X_1 + 2X_2 + 3X_3 \to \max$$

under the restriction

$$4X_1 + 3X_2 + X_3 \leq 10$$

$X_{1,2,3}$ are integers and non negative.

Suppose that the value of X_1 is searched for the first, and then the values of X_2 and X_3 are defined. Let us investigate the latter step consisting of the selection of the value of X_3. In our further considerations we shall refer to the following table given below

Table 3.2

X_3	L_3	$\psi_{1,2}$
0	$X_1 + 2X_2$	$4X_1 + 3X_2 \le 10$
1	$X_1 + 2X_2 + 3$	$4X_1 + 3X_2 \le 9$
2	$X_1 + 2X_2 + 6$	$4X_1 + 3X_2 \le 8$
	...	
n	$X_1 + 2X_2 + n$	$4X_1 + 3X_2 \le 10 - n$

According to Bellman's principle one may write

$$L_3^* = \max_{X_3}\{L_2 + 3X_3\} = \max_{\psi_{1,2}} L_3 .$$

Consider, for instance, the rows corresponding to $X_3 = 0$ and $X_3 = 1$. We obtain two particular problems as follows:

a) $X_1 + 2X_2 \rightarrow \max$

$4X_1 + 3X_2 \le 10$

$L_{2,a} = X_1 + 2X_2$

b) $X_1 + 2X_2 + 3 \rightarrow \max$

$4X_1 + 3X_2 \le 9$

$L_{2,b} = X_1 + 2X_2 .$

Compare now $L_{2,a}$ and $L_{2,b}$. To do this ,let us take into account the fact that the restriction in particular problem a) is weaker (looser) than in problem b), i.e.

$$\psi_a: 4X_1 + 3X_2 \le 10$$
$$\psi_b: 4X_1 + 3X_2 \le 9.$$

In problem a) one has the possibility to use an additional unit with the aim of increasing the value of $L_{2,a}$, i.e. it is correct to write

$$L_{2,a} = L_{2,b} + \Delta L_{a,b},$$

where $L_{a,b}$ is an increment of the functional L_3^* due to an appropriate reassignment of the additional unit between the unknown values X_1 and X_2 . It is easy to see, that the maximal possible value of $\Delta L_{a,b}$ is equal to 2/3 (due to increasing the value of X_2 by 1/3). This consideration results in finding the optimal value of X_3 which is equal to 9 (since the value of $\Delta L_{a,b}$ cannot be fractional). With the optimal value of $X_3^* = 9$, one obtains the formulation of the new task:

$$X_1 + 2X_2 \to \max$$
$$4X_1 + 3X_2 \le 1$$
$$X_{1,2} \ge 0 .$$

This problem is resolved by using the values $X_1^* = X_2^* = 0$. Thus, the final solution to the initial optimization problem is

$$X_1^* = X_2^* = 0;\ X_3^* = 9.$$

Thus we have shown that the "greedy" strategy in its pure form is applicable if, at each step, it does not lead to loss of the global optimal solution. We have reestablished the well-known Bellman's tenet having remarked that choosing the solution at the final step in accordance with this strategy did not lead to loss of an optimal solution. "Separating" the final step from the whole procedure, we obtain a new problem of a smaller size, but this new problem appears to be dependent of the last choice of X_N. Finding such a dependancy may be not a simple task or may even be impossible. It may be significantly simplified in the case of a small and discrete set of all possible variants, as in the shortest-path problem, for example. In addition, a criterion of optimality in the problems considered above was a kind of additive function of the independent subcriteria which may be considered as an essential constraint on the whole strategy, but this question is beyond our interests in this book.

3.7.7. Principle of similarity

It is assumed that the analogy between two systems (objects) is based on existing fundamental laws. Evidently, if there is an isomorphism between two problems as formal systems then the solution of each of them is suitable for the other and vice versa. Generally speaking, the goal of finding direct analogies between problems is mainly explained by the searching for a more evident interpretation of a given problem. The evidency of such an interpretation may be quite conditional and in any case is human-dependant.

Natural interest is aroused concerning the approximate analogy between two problems or their similarity. The question of the applicability of one problem solution to another is considered, for example, in [54]. Following this general idea, consider the illustration outlined in [55].

Let us consider the scheduling problem in a homogeneous multiprocessor system with $n \geq 1$ processors. Let us denote the set of processes by $T = \{T_1, T_2, \ldots, T_m\}$, where T_i is an individual process with processing time τ_i. Let $\theta = \{\tau_i\}, i = \overline{1,m}$ be the set of all processing times. Suppose further, that there is a partial order ($\succ$) representing a precedence relation between processes such that $T_i \succ T_j$ if process T_j may not be initiated before the completion of process T_i. Let us also require for $\succ$ to be irreflexive and asymmetrical relation. In this case the pair $\langle T, \succ \rangle$ may be interpreted by an oriented graph $G(T, \overline{U})$ with vertex set T and arc set $\overline{U}$ such that $\left(\overline{T_i, T_j}\right) \in \overline{U}$ if $T_i \succ T_j$ and $\bar{\exists} k(T_i \succ T_k \succ T_j)$ (Fig. 3.25). We admit that every process T_i may be interrupted at an arbitrary time in its execution and be renewed afterwards.

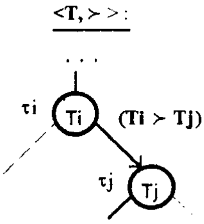


Fig. 3.25.

There are also given the due dates $D = \{d_1, d_2, \ldots, d_m\}$ corresponding to processes $T_1, T_2, \ldots T_n$ which impose time restrictions on the completion times of the processes f_i, that is,

$$f_i \le d_i, i = \overline{1, m}.$$

To summarize the all above, the model of the scheduling problem under discussion may be represented by a 5-tuple of the form

(a) $\langle T, \succ, n, \theta, D \rangle$,

where n is the number of processors in the system.

Problem definition. Let $f^{\max} = \max(f_1, \ldots, f_m)$. If some element of the 5-tuple (a) is not defined then it will be denoted by an asterisk (*). Consider the following model:

(b) $\langle T, \succ, n \ge 2, \theta, D = * \rangle$.

With respect to this model, let us formulate the problem "**PRA**" as follows:

"To find a schedule with interruptions providing

$$\min f^{\max}.$$

To solve the "**PRA**"-problem we should solve the "**PRB**"-problem, corresponding to the model

(c)$\langle T, \succ = *, n = 1, \theta, D \rangle$.

The notation "$\succ = *$" is used to indicate that the release dates of the processes are undefined (i.e. are arbitrary and may not coincide)."

"**PRB**"-problem: " Given model (c), find a scheduling strategy providing (if it is possible) $f_i \le d_i$ for all $i = \overline{1, m}$."

Let us now use the solution to the **PRB**-problem for solving the **PRA**-problem.

<u>Solution of the **PRB**-problem</u>. An algorithm for solving the PRB-problem is connected to the following events:

1) achieving the nearest interruption point assigned to some process T_j under execution;

2) initiating some new process.

We shall use the designation $I_{j\mu} = \left[\Delta_{j\mu}, d_{j\mu}\right]$ referring to process $T_{j\mu}$ with due date $d_{j\mu}$ and $\Delta_{j\mu} = d_{j\mu} - \tau'_{j\mu}$, where $\tau'_{j\mu}$ is the remaining processing time of the process $T_{j\mu}$ at the time $t_\mu \ge 0$. It is assumed that the times $\Delta_{j\mu}$ and $d_{j\mu}$ do not belong to the interval $I_{j\mu}$.

If all intervals $I_{j\mu}$ do not have common subintervals (see Fig. 3.26a) then the scheduling policy we are interested in is defined by the sequential processes $T_{1\mu}, T_{2\mu}, \ldots, T_{\beta\mu}$ in the order of non decreasing values $\Delta_{j\mu}$:.

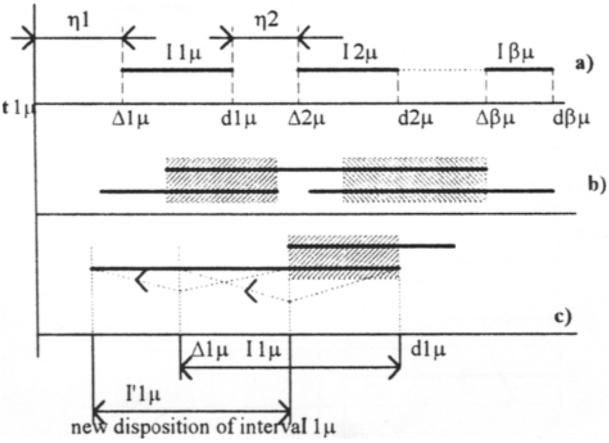


Fig.3.26

In the case of some intersecting intervals $I_{j\mu}$ (Fig. 3.26b) one has to dispose of the common subintervals (which are dashed in Fig. 3.26b,c) in the free spans denoted by η_i (see fig 3.26a).Any process $T_{\alpha\mu}$ is permitted to use only those free spans which are situated to the left of the point $\Delta_{\alpha\mu}$. The feasibility of a schedule is determined, as one can see, by the total magnitude of the free spans η_i.This magnitude should be sufficient to remove all common subintervals.

The common rule is the following:

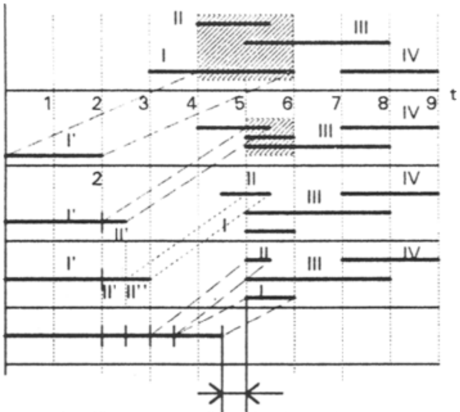
A. Choose an interval $I_{1\mu}$ with minimum value $\Delta_{1\mu}$ (or having the earliest due date $d_{1\mu}$ if there is more than one such interval).

B. If interval $I_{1\mu}$ does not intersect any other interval $I_{j\mu}$, then process $T_{1\mu}$ will be assigned to the processor with interrupt moment $t_{\mu'} = t_\mu + \tau'_{1\mu}$ (one should repeat the sequencing of remaining processes starting with the calculated time $t_{\mu'}$).

If the interval $I_{1\mu}$ has stroked subintervals, then choose the leftmost one (let its length be $L_{1\mu}$). The process $T_{1\mu}$ is assigned to processor on the interval $\left[t_\mu, t_{\mu'}\right]$, where

$$t_{\mu'} = \min\left(\Delta_{2\mu}, t_\mu + L_{1\mu}\right).$$

To illustrate this scheduling policy we give an example in Fig. 3.27.



insufficient span to fit stroked interval

Fig. 3.27.

Corollary 1. The feasibility of schedule (Sh) for the remaining processes would not be violated if Sh is feasible before the assignment of process $T_{1\mu}$ with a minimum value $\Delta_{1\mu}$ on interval $\left[t_{\mu}, t_{\mu'}\right]$ where $t_{\mu'}$ is defined according to the common rule formulated above.

Proof.

1). If interval $I_{1\mu}$ does not intersect other intervals then the total length of the free spans situated to the left of the time $\Delta_{2\mu}$ will be the same despite the occupation of interval $\left[t_{\mu}, t_{\mu'}\right]$ by the process $T_{1\mu}$.

2). If interval $I_{1\mu}$ intersects other intervals then to make good the time deficit of the process $T_{1\mu}$, defined by the stroked part of $I_{1\mu}$, one may use only those free spans $\eta_x^{(1)}$ which are situated to the left of the point $\Delta_{1\mu}$. There is the only way to achieve this, that is, to dispose of the stroked part of the interval $I_{1\mu}$ within the boundaries of the intervals $\eta_x^{(1)}$ (see Fig. 3.26 c). As a result, the interval $I_{1\mu}$ will occupy a new position where it does not intersect any other interval (Fig. 3.26c) and point (1) of this proof now may be applied.

Evidently, if the relation $\Delta_i \le \Delta_j \leftrightarrow d_i \le d_j$ holds for every i, j then one may set a preemption point for the process $T_{1\mu}$ at time

$$\omega_{\mu+1} = \min\left(\Delta_{2\mu}; t_{\mu} + \tau'_{1\mu}\right).$$

In the general case, $\omega_{\mu+1}$ is chosen with regard to the length $L_{1\mu}$ of the stroked part of an interval $I_{1\mu}$ only, i.e.

$$\omega_{\mu+1} = \min\left(\Delta_{2\mu}; t_{\mu} + L_{1\mu}\right).$$

Now we can consider the "**PRA**" - problem solution. The concept of making use of the previous solution to the " **PRA**" - problem is as follows:

- to define due dates $d_1, d_2, \ldots, d_m$ for the processes $T_1, T_2, \ldots, T_m$ in some optimal way (to be clarified later);

- to use the fact, that the above outlined scheduling policy assumes that $\succ = *$ and, therefore, is suitable for any concrete form of precedence relation $\succ$. ..

Lemma 3.1. For a given number n of processors it is correct to say that

$$L(Sh) = \max_i\{f_i\} \geq \frac{\sum_{i=1}^{m} \tau_i}{n},$$

where L(Sh) denotes the schedule Sh length, and

$$\frac{\sum_{i=1}^{m} \tau_i}{n} = \inf f^{\max}.$$

With regard to $\inf f^{\max}$, one can set such due dates which, if satisfied ,will provide

$$\min(L(Sh) - \inf f^{\max}).$$

A procedure for defining optimal due dates (from now on denoted as the D-procedure) is as follows.

1) If for all processes $T_{\varepsilon\gamma}$, such that $T_\varepsilon \succ T_{\varepsilon\gamma}$, there are known the corresponding values of $\Delta_{\varepsilon\gamma} (\gamma = 1, 2, \ldots)$ then assume

$$d_\varepsilon = \min_\gamma (\Delta_{\varepsilon\gamma})$$

and $\Delta_\varepsilon = d_\varepsilon - \tau_\varepsilon$

else go to step 2.

2) For the processes T_x having no succesors in the precedence graph assume

$$d_x = \inf f^{\max}$$

$$\Delta_x = d_x - \tau_x.$$

Consider the illustration in Fig. 3.28. In this case we have:

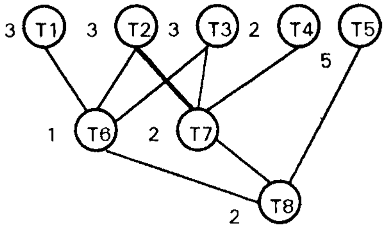


Fig. 3.28.

$$\inf f^{\max} = \frac{3+3+3+2+5+1+2+2}{3} = 7$$

$$d_8 = 7; \Delta_8 = 7-2 = 5;$$

$$d_7 = d_6 = 5; \Delta_6 = 4; \Delta_7 = 3;$$

$$d_2 = \min(\Delta_6, \Delta_7) = 3; \Delta_2 = 0;$$

$$d_1 = 4; \Delta_1 = 1;$$

$$d_4 = 3; \Delta_4 = 1;$$

$$d_3 = \min(\Delta_6, \Delta_7) = 3; \Delta_3 = 0;$$

$$d_5 = 5; \Delta_5 = 0.$$

Note. The D-procedure even allows negative values of optimal due dates. In these cases one should create a schedule starting from the time point corresponding to the negative value Δ_x with the highest absolute meaning among all the negative values Δ_i. The resulting schedule in this case should be corrected by adding the value $|\Delta_x|$ to all time moments found.

Let us start the sequencing processes on the basis of the optimal due dates. The complete scheduling process is divided into two stages. At the first stage we create a schedule for the optimal number of processors which may not coincide with those given in the "**PRA**"-problem model. It is obvious that there exists a number n_{opt} of processors which, if increased, does not lead to an improvement of the criterion

$$\min\left(L(Sh) - \inf f^{\max}\right).$$

At the second stage we must correct the resulting schedule in accordance with the given number of processors.

Let $\phi'^{\mu} = \{T_{1\mu}, T_{2\mu}, \ldots, T_{\beta\mu}\}$ be an ordered set where

$$\Delta_{1\mu} = \left(d_{1\mu} - \tau'_{1\mu}\right) \leq$$

$$\Delta_{2\mu} = \left(d_{2\mu} - \tau'_{2\mu}\right) \leq$$

$$\ldots$$

$$\Delta_{\beta\mu} = \left(d_{\beta\mu} - \tau'_{\beta\mu}\right)$$

at time t_{μ}. Let the number of processes for which $\Delta_{i\mu} = t_{\mu}$ be $n_{\mu}\left(n_{\mu} \in \{0,1,\ldots,\beta\}\right)$ and the number of all available processors is $n \geq 2$. Then

1) If $n \geq n_{\mu}$ then the first n processes in ϕ'^{μ} are assigned to the n processors with the same interrupt point at time $\omega_{\mu+1}$.

$$\omega_{\mu+1} = \min\left(\omega_{1\mu}, \omega_{2\mu}, \ldots, \omega_{n\mu}\right),$$

where $\omega_{i\mu}$ - is the time of the interrupt point assigned to process $T_{i\mu}$ accordingly to the common rule in the "**PRB**"-problem without taking into account other processes from $\{T_{1\mu}, T_{2\mu}, \ldots T_{n\mu}\}$.

2) If $n < n_{\mu}$ then additional $(n_{\mu} - n)$ fictional processors are introduced that enables us to use point (1) of the given algorithm, which is further referred to as the SR-algorithm ("Service-on-Ready"). It is also assumed that at the point $\omega_{\mu+1}$, the real number of n processors is to be restored.

Consider again Fig 3.28. An initial interval disposition is shown in Fig. 3.29a. At time $t_{\mu} = 0$ there are processes T_2, T_3, and T_5 with $\Delta_2 = \Delta_3 = \Delta_5 = 0$.

The processes T_2, T_3, T_5 are assigned to processors with interrupt point at time $\omega_1 = 1$ defined from the relation below:

$$\omega_1 = \min_i\left(\min\left(t_{\mu} + L_{i\mu}; \Delta_1\right)\right), i = 2,3,5.$$

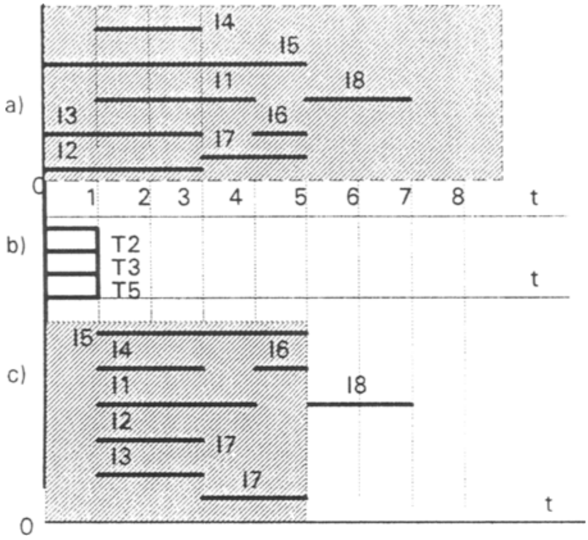


Fig.3.29 a) b) c)

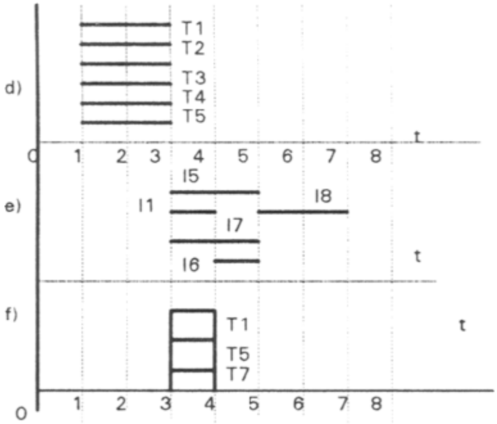


Fig 3.29 d) e) f)

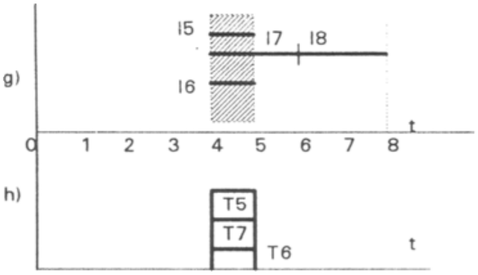


Fig.3.29 g) h)

Here $L_{i\mu}$ is the length of the leftmost stroked part of an interval $I_{i\mu}$.

At time $t_1 = 1$ $\phi^1 = \{T_3, T_5, T_1, T_2, T_4\}$. Since $n_\mu \geq n$, we have to introduce 2 additional fictional processors (Fig.3.29d). An interruption point is now assigned at time $t_2 = 3$. The disposition of the intervals for this time is shown in Fig.3.29e. $\phi^2 = \{T_1, T_7\}$. Since $n > n_\mu$, the process T_5 is also assigned to a processor. Further steps are performed by analogy and are illustrated in Fig.3.29 f, g, h. Optimality of the SR-schedule is defined by the fact that all processes complete on time according to the D-procedure. We now realize the second stage of our scheduling strategy under the assumption that $n \geq 1$ fictional processors were used. The reader is referred to [55] for an algorithm providing the mapping of intervals with n_1 processors to the optimal length intervals with n_2 processors. Fig.3.30 illustrates this algorithm by mapping the interval [1,5] with n = 5 processors to the interval $\left[1, 7\frac{2}{3}\right]$ with n = 3 processors. Thus, we obtain the resulting schedule L(Sh) = 8 $\frac{1}{3}$ [55].

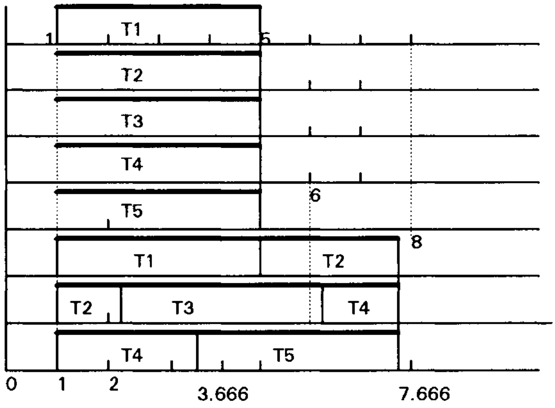


Fig. 3.30

3.7.8. Transforming the conditions of the problem

Sometimes it is possible to loosen some restrictions on the task domain provided the domain is modified appropriately in order to retain the equivalence between the new problem and the old one. Let us, for example, consider the problem of the maximum matching set in a graph. Let there be given a graph G(U,V) with a set of vertices U and a set of weighted edges V. Each eadge V_{ij} is assigned with T_{ij}. It is required to find a set $V^* \subset V$ with maximum total weight such that no two edges from V^* have common vertices. The problem may be interpreted on a weighted 0,1-matrix B with elements $b_{ij} = 0$, if edges i and j have no common vertex (-ices), and $b_{ij} = 1$ otherwise. This is quite similar to the maximum zero submatrix problem except that there are integer weights assigned to the matrix rows. This last condition may be considered as an additional restriction. For instance, let B take the form

	V_1	V_2	V_3	V_4	V_5	C
V_1	0	1	0	0	1	2
V_2	1	0	0	1	1	3
V_3	0	0	0	0	1	1
V_4	0	1	0	0	0	2
V_5	1	1	1	0	0	1

Then, transform matrix B by replacing row i with C_i new rows identical with row i. Denote for every i these new rows by $V_{i1}, V_{i2}, \dots, V_{ic}$. It is assumed, that for every a and b B[a,b] = 0. Taking into account the above considerations, we have a new matrix.

Lemma. If row V_{ij} belongs to an optimal matching set V^* then all the rows identical with V_{ij} belong to V^* also.

The proof is self-evident. What we have now is already familiar to the reader. Thus, we have managed to eliminate an initial restriction on the task domain and get an equivalent problem.

	V_{11}	V_{12}	V_{21}	V_{22}	V_{23}	V_3	V_{41}	V_{42}	V_5
V_{11}	0	0	1	1	1	0	0	0	1
V_{12}	0	0	1	1	1	0	0	0	1
V_{21}	1	1	0	0	0	0	1	1	1
V_{22}	1	1	0	0	0	0	1	1	1
V_{23}	1	1	0	0	0	0	1	1	1
V_3	0	0	0	0	0	0	0	0	1
V_{41}	0	0	1	1	1	0	0	0	0
V_{42}	0	0	1	1	1	0	0	0	0
V_5	1	1	1	1	1	1	0	0	0

3.8. Principle of dominance and choice function

The essence of a solving procedure may be interpreted in terms of choosing alternatives from the sets of solving operators. Let $\varphi(\alpha_1,\dots,\alpha_n)$ be a choice function which is used for selecting some alternative(-s) α_i from the set $\{\alpha_j | j=\overline{1,N}\}$. The choice function φ is viewed in connection with the criterion V, which is to be optimized by the selected subset $\tilde{A}$ of the alternatives.

We shall say that an alternative α_i dominates over the set

A=$\{\alpha_j | j=\overline{1,N}\}$ if $\alpha_i \in \varphi(A)$. The set A on which $\varphi(A) \neq \varnothing$ is said to be regular. On the contrary, if φ(A) is undefined on a given set A, then A is called nonregular.

Let us call a problem $S^{(k)}$ - homogeneous, if all subsets of set A with cardinal number k are regular.

Denote by $\left|S_i^{(k)}\right|$ - the number of those subsets of the given set A with cardinal number k where φ chooses α_i.

Let us suppose that A is nonregular and $S^{(k)}$ - homogeneous. The idea of finding the solution $\tilde{A}$ optimizing the criterion V consists of the following:

(i) to operate with the regular subsets of A which have cardinal number k;

(ii) to define the values $\left|S_i^{(k)}\right|$ for the alternatives;

(iii) to use these values $\left|S_i^{(k)}\right|$ for selecting $\tilde{A}$ which will also be referred to as a probably-optimal-solution (**p.o.s.**).

Denoting by K - the number of all possible solutions;

by M($\tilde{A}$) - the number of all subsets of a set A which are better, in the sense of a criterion V, than subset $\tilde{A}$, introduce an aposterior estimation

$$p^{(pos)} = \frac{K - \tilde{M}(A)}{K}$$

of a probability for the subset $\tilde{A}$ to be an optimal solution.

One can write an expression for the probability P_i of including the alternative α_i in the optimal solution in the form:

$$P_i = \frac{K_i}{K} \cdot \sum_{j=1}^{K_i} P_j^{(p.o.s.)}(i),$$

where K_i is the total number of valid solutions containing α_i;

$P_j^{(p.o.s.)}(i)$ is the probability of including α_i in the j-th **p.o.s**.

The formal explanation of the **p.o.s** may be done on the basis of the following inductive scheme. The idea is that of showing that including an object (alternative) in **p.o.s** keeps all relations (> / =) between $|S_i|$- degrees corresponding to the remaining alternatives. The selection of alternatives is performed in accordance with the nondecreasing order of their $|S_i|$-degrees. Choose alternatives until there is only one alternative to be selected from among remaining ones. For this last alternative the adequacy of its $|S_i|$-degree and the corresponding probability P_i, is evident (as it dominates over the remaining alternatives and, consequently, has a maximum value of its $|S_i|$-degree and probability P_i).

However, let us, suppose that this scheme breaks down the correspondence between the $|S_i|$-degrees and the probabilities P_i. Then the last alternative has the maximum value of a $|S_i|$-degree but not the maximum value of probability P_{i}. This supposition, however, contradicts the definition of the $|S_i|$-degrees. Let us now show that after choosing an alternative $\boldsymbol{\alpha_z}$ under the supposition that $|S_z| \geq |S_x| \geq |S_y|$ for some alternatives $\alpha_z, \alpha_x, \alpha_y$, the relation

$$|S_x'| \geq |S_y'|$$

for alternatives α_x, α_y will remain true (in a probabalistic sense). We shall use the following designations:

$|S_z|(|S_x|,|S_y|)$ - the number of the valid solutions where φ has chosen $\boldsymbol{\alpha_z(\alpha_x,\alpha_y)}$.

If a solving subset $\tilde{A}$ has cardinality k, then assume $K\ = C_{|A|}^k$ to be the number of all possible solutions and

$$P_i = \frac{|S_i|}{K}$$

is a probability of choosing $\boldsymbol{\alpha_i}$ in solution. Since, $|S_z| \geq |S_x| \geq |S_y|$ is supposed to be true, then, obviously, $P_z \geq P_x \geq P_y$.

Let $|S_{x\langle z \rangle}|$ be the number of sets, containing $\boldsymbol{\alpha_z}$where φ chooses $\boldsymbol{\alpha_x}$. Thus,

$$|S_{x<z>}| = N_z \cdot P_x$$

where N_z is the total number of sets, containing $\boldsymbol{\alpha_z}$.

Similarly,

$$|S_{y<z>}| = N_z \cdot P_y \ .$$

After including $\boldsymbol{\alpha_z}$ in the required solution, $|S_x|$ and $|S_y|$ become equal to

$$|S'_x| = |S_x| - |S_{x<z>}|,$$

$$|S'_y| = |S_y| - |S_{y<z>}|.$$

Further, we have

$$|S'_x| = |S_x| - N_z P_x = (K - N_z) P_x,$$

$$|S'_y| = |S_y| - N_z P_y = (K - N_z) P_y.$$

Hence, one can see that $|S'_x| \geq |S'_y|$. Note, that if φ chooses α_x and α_z simultaneously (α_y and α_z simultaneously) then

$$KP_x - N_z P_x \geq KP_y - N_z P_y.$$

Consider two illustrations.

Example 1. Let there be given a quadratic matrix $[t_{ij}]$ with elements t_{ij} defining the expenses connected with executing job i on processor j. It is required to assign every job j to (only one) processor in such a way that every job is assigned to some processor and there are no processors executing the same job. The assignment should provide the minimum value of the total expenses.

For matrix $[t_{ij}]$

$$[t_{ij}] = \begin{array}{c|ccccc} & \underline{1} & \underline{2} & \underline{3} & \underline{4} & \underline{5} \\ 1 & 2 & 3 & 6 & 4 & 3 \\ 2 & 5 & 1 & 3 & 8 & 7 \\ 3 & 4 & 4 & 6 & 3 & 4 \\ 4 & 5 & 5 & 5 & 3 & 3 \\ 5 & 2 & 4 & 1 & 2 & 6 \end{array}$$

we shall consider S-subsets representing submatrices of dimension 2*2 (one of them is designated with dotted line). It is obvious that the choice function is defined on every submatrix with dimensions 2*2 (it chooses a diagonal pair of elements with the minimum total sum). Considering all such submatrices, we can deduce for every element its $|S_i|$-degree, representing the number of 2*2 submatrices where the choice function selects that element. Omitting unnecessary details, let us show the $|S_i|$-degrees of the matrix elements as encircled values in the corresponding cells:

$|S_i|$:

	1	2	3	4	5
1	⑫ 2	⑫ 3	② 6	⑥ 4	⑫ 3
2	⑦ 5	⑮ 1	⑫ 3	① 8	⑧ 7
3	⑦ 4	⑩ 4	⑥ 6	⑩ 3	⑩ 4
4	④ 5	⑤ 5	⑧ 5	⑭ 3	⑬ 3
5	⑪ 2	④ 4	⑮ 1	⑪ 2	① 6

Let us now take advantage of the $|S_i|$-degrees with the help of the next solving scheme: Exclude elements t_{ij} with the minimum value of $|S_{ij}|$-degree in a step-by-step fashion.

This scheme results in finding an optimal solution: $\{t_{11}, t_{22}, t_{34}, t_{45}, t_{53}\}$. The reader may obtain the same result by using the well-known Hungarian-algorithm [56]. Note, that our scheme suggests a probably-optimal-solution which may be not optimal, but has a good chance of being optimal.

Example 2. Let us consider the minimum-size cover problem with the following actual representation of a 0,1-matrix

$$M = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline f_1 & 0 & 0 & 1 & 0 & 0 & 1 \\ f_2 & 1 & 1 & 0 & 0 & 0 & 1 \\ f_3 & 0 & 1 & 1 & 0 & 1 & 0 \\ f_4 & 0 & 0 & 0 & 1 & 1 & 0 \\ f_5 & 0 & 0 & 1 & 1 & 0 & 1 \\ f_6 & 1 & 0 & 0 & 1 & 0 & 0 \\ f_7 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}$$

Using this matrix one can represent the initial covering problem as a boolean equation of the following form

$$F \equiv (f_2 \vee f_6 \vee f_7) \& (f_2 \vee f_3) \& (f_1 \vee f_3 \vee f_5) \& (f_4 \vee f_5 \vee f_6) \&$$
$$\& (f_3 \vee f_4 \vee f_7) \& (f_1 \vee f_2 \vee f_5) \equiv 1$$

where it is required to find a solution providing the minimum of $(f_1 + f_2 + \ldots + f_7)$ ($f_i = 1$, if row i is included in the solution, and $f_i = 0$, otherwise). Let us write down all the disjunctions from F, i.e.

$$G_1 = (f_2 \vee f_6 \vee f_7)$$
$$G_2 = (f_2 \vee f_3)$$
$$G_3 = (f_1 \vee f_3 \vee f_5)$$
$$G_4 = (f_4 \vee f_5 \vee f_6)$$
$$G_5 = (f_3 \vee f_4 \vee f_7)$$
$$G_6 = (f_1 \vee f_2 \vee f_5).$$

Consider the domain $A = \{G_1, G_2, \ldots, G_6\}$. Let the choice function be defined on the pairs $< G_i, G_j >$ as follows:

$$\varphi(G_i, G_j) = \{f_x, f_y, \ldots\}$$

if and only if

$f_x \in G_i \& f_x \in G_j$

$f_y \in G_i \& f_y \in G_j$

. . . (etc).

$\varphi(G_i, G_j) = \varnothing$, otherwise.

For instance, $\varphi(G_1, G_2) = \{f_2\}$

$\varphi(G_5, G_6) = \varnothing$.

We shall find the $|S_i|$-degrees of the alternatives $f_1, f_2, \ldots, f_7$ by counting all the pairs $< G_i, G_j >$ $(x \neq y)$ where φ chooses f_i.

Thus, we obtain the following values:

$|S_{f1}| = 1; |S_{f2}| = 3; |S_{f3}| = 3; |S_{f4}| = 1;$

$|S_{f5}| = 1; |S_{f6}| = 1; |S_{f7}| = 1$.

Let us delete the row with the minimum $|S_i|$-degree, for example, f_1, and then, having corrected the $|S_i|$-degrees, repeat this procedure again until one or more rows are deterministically included in the solution. This process results in the following p.s.o.: $\{f_2, f_3, f_6\}$ which in turn is one of the minimum-size covers.

3.9. An example of mechanization of heuristics

As was pointed out earlier, one needs to formalize a given heuristic(s) in order to apply it to an actual problem. We have considered a number of problems, where such a formalization was human-made. It is quite natural to suppose that the mechanization of heuristics should require the creation of a special theory which we do not intend to develop here. Instead, we will address ourselves to the application of enhanced traditional methods.

Obviously, the very notion of a heuristic is rather fuzzy, therefore, it is both context and human-dependant.

Heuristic, as we understand it, does not indicate real actions but merely defines their character, i.e. it is a condition-action frame. This frame may be represented by a 2-tuple:

$$\Pi = \langle C, A \rangle,$$

where C is a condition frame;

A is an action frame.

It should be noted that П merely distinguishes between a concrete condition "c" and a concrete action "a", i.e. for given "c" and "a" one may say that if "c" satisfies C then it may provoke any action "a", satisfying A. To be more precise, we shall associate C and A with vectors $s = \langle s_1, s_2, \dots, s_n \rangle$, where each component $s_i \in \{0, 1, *, 2\}$.

Let us now devise an interpretation for s. If $s_i = 0$ in C then it is interpreted as "absent" for some condition (-s); if $s_i = 1$ then a corresponding condition is considered to take place; if $s_i = *$ in C then it is not important in respect to the given heuristic P; if $s_i = 2$ then the real value of s_i is undefined. By analogy, for the action frame A, we establish the following interpretation: if $s_i = 0(1)$, then applying any action satisfying C results in setting the component s_i of state-vector **s** to 0(1); if $s_i = *$, then no action satisfying C may change the current value of s_i in **s** and, finally, if $s_i = 2$ then the resulting value of s_i may be either 0 or 1.

Let us give an explanatory example. Consider the following heuristic

$$П=\langle C = \langle 01 * * \rangle; A = \langle 210 * \rangle \rangle.$$

According to the given specification of C, one may conclude that in order to apply any action "a" satisfying A it is necessary that, in the state-vector $s = \langle s_1, s_2, s_3, s_4 \rangle$ standing for C, s_i should be equal to 0 and s_2 should be equal to 1. The values of s_3 and s_4 can be arbitrary.

The action frame A in the above example may be extended to the set of real actions satisfying A, i.e. to the following set:

$$\begin{Bmatrix} 110* \\ 010* \end{Bmatrix}.$$

Note that component s_4 in both vectors is equal to "*" as they cannot change it by definition. Suppose now that the current state-vector **s** is <0110>. It is easy to see that our heuristic Π is valid in **s** as a condition combination in C covers this individual vector. It is, therefore, clear that applying the general action A will result in a new state-vector of the form <2100>. For another current state-vector <0212> we obtain the new state-vector in the form <2102> ,etc.

Suppose now that the current state-vector is <2102> and it was directly obtained by applying the heuristic above. Then one may deduce that the previous state-vector was <0122>. Comparing this result with that obtained above, one may see some discrepancy in the specification of the same vector from which vector-state <2102> has been obtained.

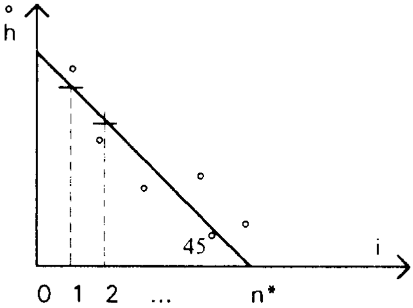
Thus, our goal is to employ a traditional A^* - based approach in order to remain within the boundaries of a given problem specification (i.e. $<S^0, S^f, \{\Pi_i\}>$). More precisely, we are going to introduce some new techniques based on a traditional ideology of heuristic evaluation function utilization. As the reader can note from our particular heurictic specification, the problem of the mechanization of heuristics is reduced to a partly defined problem.

Consider a state-tree T of a given problem with the set of nodes S standing for corresponding state-vectors S_i . Let us denote the nodes as a,b, ... or $S_a, S_b, \ldots$. For a given node "a" we have a A^* -based heuristic evaluation function f(a) of the form

f(a) = g(a) + h(a),

where $h(a)$ evaluates the length of an optimal path in T connecting "a" to the final state S^f Let us answer the question: "does "a" really belong to the optimal path P^{opt} connecting S^0 (initial state) to S^f?". Suppose, that the function h is defined exactly. Then it is clear that if node "a" belongs to P^{opt} and node "b" is a direct successor of node "a" then "b" belongs to P^{opt} if and only if h(b) = h(a) - 1.

This simply means that from node "b" one needs to open subsequently as many nodes as from node "a" subtracted less one. Denoting by $\overset{0}{h}(i)$,the value of the function h for the number i of opened nodes, an ideal graphic of $\overset{0}{h}$ may be represented as shown in Fig. 3.31.



n* is an optimal path length

Fig.3.31

Evidently, $\overset{0}{h}(0) = n^*$. As the real value of h(x) is unknown one should use an estimate $\hat{h}(x)$ instead of $h(x)$. This leads to an "arbitrary" disposition of points around the "ideal line" (in Fig. 3.31 these points are denoted by circles). It is clear that for any path in T, emanating from S^0, the corresponding set of points may be obtained on the basis of $\hat{h}(x)$. For example, in Fig. 3.32 two different sets of points are shown for the imaginary paths P_1 and P_2 .

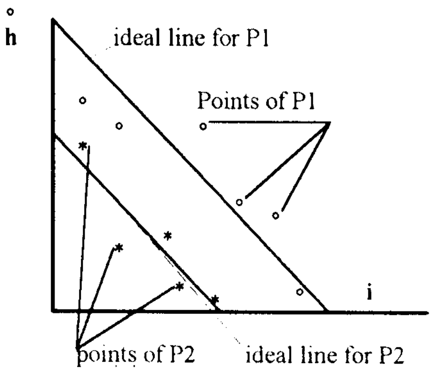


Fig 3.32

The following observation is essential for further considerations, namely: "in the supposition that the given set of points corresponds to an optimal path P^{opt} ,there is the only "ideal line" approximating this set of points". To obtain this "ideal line", consider a set of points represented by the pairs

$$(x_1, y_1),(x_2, y_2),\dots,(x_n, y_n).$$

An approximating line of the form

y = ax + b

may be found on the basis of well-known criterion

$$\sum_{i=1}^{n}\left(y_i-\left(ax_i+b\right)\right)^2 \rightarrow \min.$$

In our notation we have

$$\overset{0}{h}(\mathrm{i}) = \mathrm{a} \bullet \mathrm{i} + \mathrm{b},$$

which for $i = 0$ and $\overset{0}{h}(0) = 0$ gives $a = -1$. Actually,

$$\overset{0}{h}(0) = b = n^*_{opt}$$

$$\overset{0}{h}(i) = 0 \rightarrow n^*_{opt} = \frac{b}{-a}$$

So, we have

$$\sum_{i=0}^{n}\left(\overset{0}{h}(i)-(-i+b)\right)^2 \rightarrow \min.$$

By setting $\dfrac{\partial \overset{0}{h}(i)}{\partial b} = 0$ one may obtain

$$\sum_{i=0}^{n}\left(\overset{0}{h}(i)+i-b\right)=0,$$

which in turn gives

$$\left(\sum_{i=0}^{n}(\overset{0}{h}(i)+i)\right)-(n+1)b=0,$$

$$\sum_{i=0}^{n}\overset{0}{h}(i)+\frac{(n+1)n}{2}=(1+n)b,$$

and $b=\dfrac{\sum_{i=0}^{n}\overset{0}{h}(i)+\dfrac{(n+1)n}{2}}{n+1}=\dfrac{1}{n+1}\cdot\sum_{i=0}^{n}\overset{0}{h}(i)+\dfrac{n}{2}.$

Replacing i by $g(i)$ we may obtain another equation for b in the form

$$(n+1)b=\sum_{x\in Path}f(x),$$

where "Path" represents a path in T with length n which starts in S^0 and traverses (n+1) nodes.

These formulae enable one to find an estimate for n^*_{opt} as it was shown earlier that $n^*_{opt}=b$ provided the path P, with a set of values $H_p=\left\{\overset{0}{h}_0,\overset{0}{h}_1,\dots,\overset{0}{h}_n\right\}$, is an optimal one. It is essential that the result we obtained is connected to the "history" of the searching process represented by the sequence H_p on the one hand, and is not restricted by A^* - based supposition that $h(x)\geq\hat{h}(x)$ where $\hat{h}(x)$ is an estimate of $h(x)$, on the other. The only essential restriction is in the requirement for H_p to be a correlated sequence which primarily depends on the accuracy of the estimation function $\hat{h}(x)$.

Consider again Fig 3.32. Suppose that one has the statistics for a given $Path\langle i_0, i_1, i_2, \ldots, i_n\rangle$ represented by a sample $\left\langle \overset{0}{h}(i_0), \ldots, \overset{0}{h}(i_n)\right\rangle$. These statistics enable one to use the equation for "b" to find a prognosticated value of $\overset{0}{h}(i_{n+1})$. It is a well-known fact that the "real" value of $h(i_{n+1})$ falls in the interval $\overset{0}{h}(i_{n+1}) \pm 3\sigma$ (where σ is $\sqrt{D}$, and D is a deviation of a given sample) with probability 0,997. To take an advantage from the fact suppose that every next opened node provides the maximal decrease of real value $h(i_{n+k})$ in comparison with the prognosticated one. It means, that $\overset{0}{h}(i_{n+k}) = h(i_{n+k}) + \alpha_{\max}$, where $\alpha_{\max}\rangle 0, \alpha_{\max} \approx 3\sigma$. From this supposition one can derive the following important result. As it was found

$$b_n = \frac{1}{n+1} \cdot \sum_{i=0}^{n} \overset{0}{h}(i) + \frac{n}{2}.$$

$$\text{Hence, } h_{n+1} = \overset{0}{h}_{n+1} - \alpha_{\max} = -n - 1 + \frac{1}{n+1} \cdot S_n + \frac{n}{2} - \alpha_{\max},$$

$$\text{where } S_n = \sum_{i=0}^{n} \overset{0}{h}(i).$$

$$b_{n+1} = \frac{1}{n+2}\left(S_n + \left[-n - 1 + \frac{1}{n+1} S_n + \frac{n}{2} - \alpha_{\max}\right]\right) + \frac{n+1}{2}.$$

Thus, we obtain (omitting details)

$$\Delta b_{n,n+1} = b_n - b_{n+1} = \frac{\alpha_{\max}}{n+2}.$$

To proceed, let us note that Δb defines the maximal shift of the regression line provided new value of

$$h_{n+1} = \overset{0}{h}_{n+1} - \alpha_{\max}$$

exists (see illustration in Fig 3.33).

If, starting from the current point i_n, every next point $i_{n+1}, i_{n+2}, \ldots$ will be declined from a regression line as much as a value of $\alpha_{\max}$, then minimal value K which provides inequality

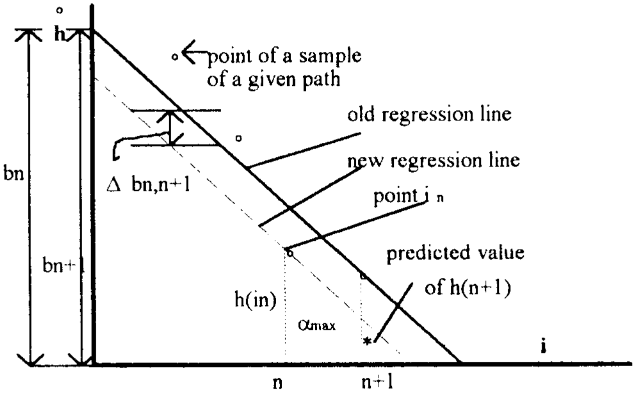


Fig3.33

$$\frac{\alpha_{\max}}{(n+1)+1} + \frac{\alpha_{\max}}{(n+1)+2} + \ldots + \frac{\alpha_{\max}}{(n+1)+k-1} \geq \frac{1}{n+1} \cdot \sum_{i=0}^{n} \overset{0}{h}(i) - \frac{n}{2},$$

defines a lower bound on the number of the nodes to be opened before reaching the final state-node S^f from i_n. It may be derived from the above that

$$\alpha_{\max}\left(1 + \frac{1}{n+2} + \frac{1}{n+3} + \ldots + \frac{1}{n+k}\right) \geq b_n - n.$$

Using an approximate equation

$$\frac{1}{n+2} + \frac{1}{n+3} + \ldots + \frac{1}{n+k} \approx \frac{k-1}{n+2} - \left\{ \frac{k-1}{n+2} \cdot \frac{\frac{1}{n+3} + \frac{k-2}{n+k}}{2} \right\},$$

and omitting all intermediate calculations one may obtain the following result:

$$k^2 \cdot [2(n+3)-1] + k \cdot [2(n+3)(n-1)+4-2 \cdot C] - 2n(n+3) - 2C \cdot n - 4 \geq$$

where $C = \dfrac{(b_n - n)(n+2)(n+3)}{\alpha_{\max}}$.

Thus, for $\alpha_{\max} = 2, b_n = 20, n = 10$ we obtain $k \approx 7$. All that remains is to integrate the results obtained.

Let there be found the values of $n_{path}, b_{npath}, K_{path}(n_{path}, b_{npath}, \alpha_{\max})$ for a given path in a state tree T. It is obvious that the minimum possible path length is

$$L_{path}^{\min} = n_{path} + K_{path}(n_{path}, b_{npath}, \alpha_{\max}).$$

Following Nilsson [5], formulate this result with the following theorem.

Theorem. A heuristic evaluation function $f(x) = g(x) + h(x)$ with arbitrary function $h(x)$,may be replaced by the function $f^*(x) = g(x) + K(x, b_x, \alpha_{\max})$,, provided that $h(x) \in \left[\overset{0}{h}(x) - \alpha_{\max}, \overset{0}{h}(x) + \alpha_{\max}\right]$, where $\overset{0}{h}(x)$ is an exact value of a minimum cost path from vertex x to the final node.

Proof. If the condition $h(x) \in \left[\overset{0}{h}(x) - \alpha_{\max}; \overset{0}{h}(x) + \alpha_{\max}\right]$ is true, then it follows that $K(x, \alpha_{\max}) \leq \overset{0}{h}(x)$. According to Nilsson [5] A^* , with heuristic function $f^*(x) = g(x) + K(x, b_x, \alpha_{\max})$, always stops at the final node with the minimum cost solution path found.

3.10. Conclusion

We summarize this section as follows. In order to improve the strategies based on heuristic evaluation function $f = g + h$,the following concepts can be realized:

(1) using a probabalistic evaluation function as in 3.9 to relax the restriction on the monotonicity of the function h;

(2) estimating the values of the function h by means of some heuristic procedure as in 3.7.1;

(3) cutting the parts of a search tree as was proposed in 3.4.3,3.7.2,3.8;

(4) evaluating the function *h* starting from the lower levels of a search tree (3.7.4);

(5) using heuristic principles to produce weak methods (3.6, 3.7).

This Page Intentionally Left Blank

Chapter 4

LOGIC-BASED PROBLEM SOLVERS: APPROACHES AND NEW METHODS

4.1. Introduction

In this chapter we give an outline of logic-based problem solvers and new theoretical methods of logical inference which possess a number of positive features. The reader should bear in mind that there are three main directions of logic utilization in **CAPSS:**

- as a problem solving technology providing theoretical basis of so-called logic-based problem solvers incorporated in **CAPSS**;

- as an intelligent controller which is used to test the correctness of task transformation steps consisting of equation generating and modification, making substitutions, inferences, etc.;

- as an intelligent oracle representing a subsystem of **CAPSS** , specializing in human-machine hypotheses making and treatment.

The second and the third points are discussed in the next chapter.This chapter deals with the first point in the above list and is organized in two parts:

the first one represents a depiction of logical problem solvers and the second one contains an outline of two new logical inference methods based on the cut principle.

4.2. Logical problem solvers

One of the interesting paradigms of solving technologies is that one connected to logical inference. This paradigm is based on proving a so-called theorem of solution existence:

$$\forall x \exists y\, \varphi(x,y)$$

which asserts that y is the ouput of a problem φ with x as its input.

Many works have been devoted to using the proving technique with the aim of solving problems with clearly stated solution existence theorem [1,2]. However, it is not always possible to state a solution existence theorem in a practically acceptable way. For example, in dynamic models, where applied operations directly lead to the alteration of a static model representation. Another area of interests is that one which uses logical means to specify a problem and a solution procedure. We will subsequently consider these aspects.

Let us start with dynamic models which are widely used in robotics and scheduling systems. In these applications the solution existence theorem gets a new interpretation. Consider an illustration of a planning problem φ(x,y) where y identifies some unknown planning sequence (which is to be found) and x denotes a pair $< X^0 ,X^{fin} >$ in which X^0 (X^{fin}) stands for the initial (final) state of the system so that y provides mapping

$$X^0 \overset{y}{\Rightarrow} X^{fin}.$$

This planning problem is rather typical for logic-based solution systems [5,48].

Let a final state X^{fin} be associated with the goal G_0 and initial state X^0 be associated with a current world model M_0. First, one should try to derive G_0 from M_0, i.e. to prove

$$M_0 \vdash G_0 .$$

It is done by classical derivation of a contradiction (□) from { $M_0 \cup \overline{G_0}$ }. If so, then a problem is trivially resolved. Otherwise, a number of disjuncts are to be defined which make it possible to derive an empty disjunct from { $M_0 \cup \overline{G_0}$ }. Let us denote these new disjuncts by D_0. Then, they should satisfy the following condition:

{ $M_0 \cup \overline{G_0} \cup D_0$ } $\vdash \square$.

Further, suitable operators are looked for which introduce these disjuncts D_0 into the world model if being applied to M_0. Clearly, all these operators should be valid in M_0, that is, their pre-conditions should not contradict M_0. Each operator is represented as a 3-tuple

Oi =<(**P**) pre-conditions;

(**LD**) list of conditions deleted;

(**LA**) list of conditions added>.

In order to apply Oi to M_0 it is needed that

$M_0 \vdash P$.

The application of operator Oi leads to deleting the conditions from LD and adding new conditions defined in LA. Formula P becomes a new goal for the planning system and all the steps relevant to G_0 are performed for this new goal. Suppose, it is proved that

$M_0 \vdash P$.

Then operator Oi is applied to M_0 which leads to a new model M_1 obtained from M_0 by deleting predicates from LD and adding predicates from LA. The procedure continues by analogy with the aim to prove $M_0 \vdash G_0$.

Consider an example. Let a world model consists of a monkey (M), a box (B) and a banana (BN). Let the model comprise the following predicates:

ATR(x) - the monkey is in the point x;

AT(BOX,y) - the box is in the point y;

$\overline{\text{HB}}$ - the monkey does not possess the banana.

Let

M_0 ={ATR(a), AT(BOX,b), $\overline{\text{HB}}$ },

G_0 =HB.

Define the following operators:

1. f1(m) - climb the box at point m;

P: ATR(m)&AT(BOX,m);

LD: ATR(m);

LA: ATR(BOX).

2. f2(m,n) - go to point n from point m;

P: ATR(m);

LD: ATR(m);

LA: ATR(n).

3. f3(m,n) - pull the box from m to n;

P: ATR(m)&AT(BOX,m);

LD: ATR(m),AT(BOX,m);

LA: ATR(n),AT(BOX,n).

4. f4 - grasp banana;

P: ATR(BOX), AT(BOX,c);

LD: $\overline{\text{HB}}$;

LA: HB.

The solution procedure consists of the following steps.

Since **not**($M_0 \vdash G_0$) is true, find D_0 = HB. Select operator f4 because HB $\in$ LA(f4). Form a new goal:

ATR(BOX)&AT(BOX,c)&$\overline{\text{HB}}$

and two new subgoals:

ATR(BOX) and AT(BOX,c).

Now we have

not(M_0 ⊢ ATR(BOX)),

not(M_0 ⊢ AT(BOX,c)).

Consider the first subgoal: ATR(BOX). Select operator f1(m) because ATR(BOX) ∈ LA(f1). A new goal becomes the same as a pre-condition of the operator f1, i.e.

ATR(m)&AT(BOX,m).

It follows then that

not(M_0 ={ATR(a),AT(BOX,b), $\overline{HB}$ } ⊢ ATR(m)&AT(BOX,m))

since m should be equal to a and b simultaneously. Thus we obtain the sets

D' = {ATR(b), AT(BOX,b)} or

D" = {ATR(a), AT(BOX,a)}.

Deleting redundant predicates, we obtain:

(a) D'= {ATR(b)}

(b) D"= {AT(box,a)}.

Choose variant (b) and select operator f3(m,a) with a pre-condition:

G_3 =ATR(m)&AT(BOX,m).

It is the same as at the previous step. Thus, to avoid repetition, choose a new subgoal ATR(b) and find the next suitable operator f2(m,b) with a pre-condition ATR(m). This time we have

M_0 ⊢ ATR(m)

for m = a.

Thus, one can apply operator f2(a, b) to M_0 and obtain a new model M_1 = {ATR(b), AT(BOX,b), $\overline{HB}$ }. Apply f3 to M_1 and obtain:

M_2 = {ATR(BOX),AT(BOX,a), $\overline{HB}$ }.

For operator fl one should provide true value of predicate AT(BOX,c). This predicate becomes a new goal and the procedure continues by analogy. We, however, omit all the corresponding details since this is a rather elementary example and makes no difficulties for the reader.

Now shift accent to the logical specification of the solving procedures which is somewhat a different way of applying logic to problem solving.

We shall deal with a scheduling problem here. First, let us specify the predicates relevant to scheduling problems.

ASG(Process,Time,Processor)

/Process is to be assigned to Processor at the moment Time/.

Dur(Process,Starttime,Finishtime)

/Process starts execution at the moment Starttime and finishes at the moment Finishtime/.

Wrt(Process,Worktime)

/Process "Process" has an execution time Worktime/.

ActProc(ListofProcesors,T)

/A list of processors which are ready to execute processes at the moment T/.

FreePr(Processor,Time)

/Processor "Processor" becomes free at the moment Time/.

Systemclock(Time)

/A system timer initially equal to 0/.

List(Listofprocesses)

/ A list of processes ready to be executed at the moment T/.

Precedence(Process1, Process2)

/ Process 2 cannot be executed before Process1 ends /.

We are interested in a scheduling policy which minimizes the total execution time of a given set of processes. Define logic of a scheduler by the specification of its components. We use the Prolog notation for the desired specification.

The following fragment is used to determine a list of processes which can be executed at the moment T:

```
findreadylist:-:
    unpreceded(X),
    list(Readylist),
    not(member(X,Readylist)),
    include(X,Readylist,ReadylistNew),
    retract(list(_)),
    assert(list(ReadylistNew)),
    fail.

findreadylist.

    unpreceded(X):-
    precedence(_,X),!,fail.

    unpreceded(_).
    member(_,[]):-!,fail.
    member(X,[X|_)]:-!.
    member(X,[_|,Y]):-!,member(X,Y).

include(X,T,[X|T]).
```

The list of ready processes is stored in the database predicate "list(Readylist)". The sense of the other predicates ("member","unpreceded") is clear from the above given clauses.

When the execution of the process is over some other process or processes could possibly become ready for the execution, i.e. the database predicate list(Readylist) should be updated. This is performed by the following fragment:

```
 refreshing:-
     systemclock(T),
     completed(X,T1),
     T1<=T,
     retractall(precedence(X,_)),
     retract(copmpleted(X,_)),
     fail.
refreshing.
```

Here, the database predicate "completed" is used to keep the completion time of the corresponding process.

The following fragment is used in order to choose a process from a list of ready processes and to assign it to a vacant processor. The vacant processors form a list ActProc.

```
update:-
  retractall(actproc(_,_)),
  systemclock(T),
  assert(actproc([],T)),
   freepr(Pr,T1),
  T1<= T,
  actproc(X,_),
  retractall(actproc(_,_)),
  include(Pr,X,Xnew),
  assert(actproc(Xnew,T)),
  fail.
update.
```

If the list of vacant processors in actproc is empty then the system clock should be modified by a predicate "changesystemclock" defined below:

```
changesystemclock:-
    retract(systemclock(_)),
    freepr(_,T),!,
    assert(systemclock(T)),
    freepr(_,T1),
    systemclock(T2),
    T1<T2,
    retract(systemclock(_)),
    assert(systemclock(T1)),
    fail.
```

It is seen from the above fragment that the system clock becomes equal to T where

T= min free(Pr,T).

If a list of ready processes is not empty then the processes are assigned to vacant processors.

```
assignment:-
    list([X|Tail]),
    actproc([Z|R],T1),
    T1<=T,
    assert(asg(X,T,Z)),
    retractall(list(_)),
    retractall(actproc(_,_)),
    assert(list(Tail)),
    assert(asg(R,T1)),
    retractall(freepr(Z,_)),
    wrt(X,WTime),
    Tfin=T+WTime,
    assert(freepr(Z,Tfin)),
    retractall(completed(X,_)),
    assert(completed(X,Tfin)),
    fail.

    assignment.
```

Now a control algorithm of our program may be represented as follows.

```
control:-
    retractall(list(_)),
    assert(list([])),
    assert(actproc([1,2,...,N],0)),
    assert(freepr(1,0)),
    assert(freepr(2,0)),
    . . . . . . . . . .
    assert(freepr(N,0)),
    repeat,
    findreadylist,
    assignment,
    changesystemclock,
    refreshing,
    update,
    not(precedence(_,_)),!.
```

The main aspect of our logic-based scheduler is connected to the question "how to implement different scheduling strategies in the main model?" Obviously, a mechanism of the specification of a scheduling policy is required together with an interpreter which translates such a specification into a logic program.

To simplify our considerations we will suppose that each scheduling strategy assignes priorities to processes in such an order that the processes with the highest priority levels are assigned to the vacant processors first. Consider, for example, the following scheduling policy:

"Processes with the largest processing time are assigned to the vacant processors first".

This policy can be interpreted as below:

```
preferrable(A,B):-
wrt(A,TA),
wrt(B,TB),
TA>=TB.
```

A more complicated variant which uses levels of processes corresponding to the nodes in a precedence graph may be represented as follows:

```
preferrable(A,B):-
    level(A,Ta),
    level(B,Tb),
    Ta>=Tb.
    level(X,Tx):-
    not(precedence(X,_)),
    wrt(X,Tx).

level(X,0):-
    retractall(workcell(_)),
    assert(workcell(0)),
    precedence(X,Y),
    workcell(Z),
    level(Y,T),
    T>=Z,
    retractall(workcell(_)),
    assert(workcell(T)),
    fail.

    level(X,T):-
    wrt(X,Tx),
    workcell(R),
    T=Tx+R.
```

Thus, the main task may be formulated in terms of finding a suitable specification language (SL) and creating a compiler from SL to Prolog.

Clearly, some kind of a mathematical language can be used as SL. For example, we could formalize the level definition as follows:

```
if not(precedence(X,_)) then wrt(X,Tx),
level_of_x= Tx
                    else
level_of_x= maximum level_of_y  + Tx
   where y ∈ {Z| precedence(X, Z)}.
```

In what follows we describe one variant of SL (all necessary details may be found in chapter 5 where a hierarchy of languages used in **CAPSS** is considered). We use designation f(...) for a function and {P} for a set P. An asterrisk right before a symbol, e.g. *X, identifies a term which is defined as an output of the corresponding specification.

f(*x1,x2) - x1 is defined through x2 in f(.);

*f(x1,x2) - the value of function f is calculated;

*f(x1,x2)=T - the value of function f is set equal to term T;

{predicate(x,*z)} - a set of all z such that predicate(x,z) is true;

{f(*x)=T} - a set of all x such that f(x)=T;

max {f(*x)=T} - maximum element in the set defined previously;

min {f(*x)=T} - see above;

for set P:

*P={a1,a2,...,an} is set P initialization;

*y∈ P y is an element of P;

{*fun({predicate(x,*y)})} - a set of values of function "fun" defined on arguments y from set {predicate(x,*y)}.

In this simplified language our definition of levels may be rewritten in the following way:

```
if not(precedence(x,_)) then
      *level(x)= wrt(x,Tx)
else
      *level(x)= max {*level({precedence(x,*y)})} + wrt(x,*Tx).
```

Elements of SL can be realized by the following Prolog fragments:

(i) ***f(x1,x2)=T**

```
assert(function(f(x1,x2),T)).
```

(ii) **{predicate(x,*z)}**

```
initialization(_):-
   retractall(predicatelist(_)),
   fail.

initialization(X):-
   predicate(X,Z),
   predicatelist(R),
   not(member(Z,R)),
   include(Z,R,R1),
   retractall(predicatelist(_)),
   assert(predicatelist(R1)),
   fail.

initialization(_).
```

As a result, a set predicatelist(Z) is obtained corresponding to

{predicate(X,*Z)}.

(iii) **{f(*x)}=T}**

Since the condition f(x)=T is not a pure logic expression it refers to an algebraic equation solution procedure.

(iv) **max {P}** where P is a set further designated as Plist.

```
choice(X):-
  Plist([R|Z]),
  select(R,Z,X).

select (R,[],R).

select(R,[Y|T],X):-
 Y>R,!,
select(Y,T,X).

select(R,[_|Z],X):-
    select(R,Z,X).
```

A special part of the interpreter deals with alternative and recursive definitions. Consider the following typical example.

```
*f(0)=T0
*f(X)= *f(*g(X-1))
```

This may be translated into the following code:

```
functiondefinition(f(X),Z):-
assert(function(f(0),T0)),
define(function(f(X),Z)),
retractall(function(_,_)).

define(function(f(0),Z):- function(f(0),Z).

define(function(f(X),Z)):-
function(g(X-1),G),
!,
define(function(f(G),Z)).
```

We have used meta-predicates such as

```
define(function(_,_))
```

which are beyond current Prolog language versions possibilities. It was done intentionally in order to avoid a lot of unnecessary details. Thus, in this section we have demonstrated the idea of using logic language (for example, Prolog) to specify solution procedures or their essential parts. A specification language is needed either to specify a problem in a formal way or to specify the rules and heuristics which are used in the solution procedure. It is clear that in this approach logic is used in a way different from its usage as a proving mechanism for a clearly stated solution existence theorem. Thus, realizing a specification language consists of the extension of knowledge processing languages and Prolog itself. Loglisp and Qute are worth to be mentioned. Qute is an amalgation of Prolog and Lisp with additional possibilities such as mechanism similar to Hilbert's ε -symbol. For example,

```
Epsilon(X;member(X,l[apple,orange]))
```

finds a value of X which makes predicate "memeber" with the definition

```
 member(X,[X|_]).
 member(X,[_|Z]):-
              member(X,Z).
true.
```

It is expected that a "mixing" Prolog and Lisp is especially fruitful for problem specification aims.

4.3. Group resolution principle in predicate calculus

4.3.1 Case of propositional system

Let us start from propositional calculus. First, let us show how to reduce DECIDABILITY PROBLEM (denoted as **DEC** for short) to MINIMUM-SIZE COVER PROBLEM (**MSCP**). Let the following system of disjuncts be given

D1=$x_1 \vee x_2 \vee \bar{x}_3 \vee x_4$

D2=$\bar{x}_1 \vee x_4$

D3=$x_1 \vee \bar{x}_2 \vee \bar{x}_4$
D4=$\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3$
D5=$x_1 \vee x_3 \vee x_{4,}$ $\qquad x_i \in \{0,1\}$.

For this system we build a covering matrix shown in Fig.4.1.

Columns D1*,...,D5* of this matrix are obtained from disjuncts D1,...,D5 respectively in the following way: " "1" is placed in row x_i ($\bar{x}_i$) and column Dj* if and only if x_i ($\bar{x}_i$) belongs to Dj. Additional columns A1, A2, A3, A4 correspond to tautologies $x_i \vee \bar{x}_i$, i=1,4.

We assert that **MSCP** defined on the covering matrix is equivalent to **DEC**-problem in the following sense: the given **DEC**-problem is resolved in a valid interpretation I if and only if the corresponding **MSCP** possesses a minimum-size cover with n rows, where n stands for a number of variables (x_i) in **DEC**-problem. Instead of a full proof we give only a sketch.

	$D1^*$	$D2^*$	$D3^*$	$D4^*$	$D5^*$	$A1$	$A2$	$A3$	$A4$
x_1	1	0	1	0	1	1	0	0	0
x_2	1	0	0	0	0	0	1	0	0
x_3	0	0	0	0	1	0	0	1	0
x_4	1	1	0	0	1	0	0	0	1
$\bar{x}_1$	0	1	0	1	0	1	0	0	0
$\bar{x}_2$	0	0	1	1	0	0	1	0	0
$\bar{x}_3$	1	0	0	1	0	0	0	1	0
$\bar{x}_4$	0	0	1	0	0	0	0	0	1

Fig. 4.1.

First, a minimum-size cover for covering the matrix contains at least n rows due to additional columns Aj. Clearly, if **DEC** is satisfied with the interpretation I then I should contain exactly n letters (otherwise, I would be contradictory). By the sense of this interpretation I, it should have at minimum one common letter with each disjunct Dj (because Dj is true in I). From this,one can directly conclude that I represents a minimum-size cover for the covering matrix.The following theorem directly refers to theorem 1.3 and corollary 2 from section 1.5.

Theorem 4.1. Let column-resolvent β be produced on the basis of column-disjuncts $D^*_{i1},\dots,D^*_{iz}$. Then disjunct β can be derived from disjuncts $D_{i1},\dots,D_{iz}$ in propositional calculus.

Proof. If π is not a minimum cover for a covering matrix corresponding to a given set of disjuncts then each minimum-size cover for this covering matrix covers β. Since a minimum-size cover corresponds to a solution of the **DEC**-problem then each feasible solution of **DEC** satisfies β. In other words, β represents a logical consequence of the disjuncts of **DEC.** These considerations remain valid with regard to the disjuncts forming a submatrix D of the covering matrix with colunns $D^*_{i1},\dots,D^*_{iz}$. Because π is not a minimum cover for D by supposition it follows that

$$\vdash D_{i1},\dots,D_{iz}\rightarrow\beta.$$

According to well-known Deduction theorem [7, 9] one can obtain that

$$D_{i1},\dots,D_{iz}\vdash\beta.$$

Thus, theorem 4.1 establishes a link between **MSCP** and **DEC**. To generalize this link to first order logic is our nearest task. From now on we shall call the principle of generation of a column-resolvent β on the basis of disjuncts $D_{i1},\dots,D_{iz}$ -group resolution principle (**g.r.p.**). Note, that it is sufficient to use exactly (n+1) characteristic column-disjuncts to produce a group resolvent for a given non-redundant cover π of a covering matrix.

4.3.2 Generalization of g.r.p. to predicate calculus

It is possible to identify a general formulation of **g.r.p.** applicable to predicate calculus. This formulation requires the following:

- A set J of column-disjuncts, participating in group resolution, should be unifiable;
- It is sufficient to use only (n+1) column-disjuncts when producing column-resolvent;
- An empty column-resolvent always testifies to a contradictory system of disjuncts.

Let us start with the following example:

C1= P(a)

C2= $\overline{D}(y) \vee L(a,y)$

C3=$\overline{P}(x) \vee \overline{Q}(y) \vee \overline{L}(x,y)$

C4= D(b)

C5= Q(b).

A covering matrix is shown in Fig.4.2.

	C_1^*	C_2^*	C_3^*	C_4^*	C_5^*	6	7	8	9
P	1					1			
D				1			1		
L		1						1	
Q					1				1
$\overline{P}$			1			1			
$\overline{D}$		1					1		
$\overline{L}$			1					1	
$\overline{Q}$			1						1

Fig. 4.2.

In this example n=4 (there are four different literals: P,D,Q,L). Let us choose five columns: $C1^*$, $C2^*$, ..., $C5^*$ and find their column-resolvent: A=∅.

Note that all the disjuncts C1,...,C5 are unifiable. Hence it follows that the system of disjuncts {C1,...,C5} is contradictory. It should be noted here that **g.r.p.** enables one to generate a resolvent of a group of n ≥ 2 disjuncts at a time. Therefore, **g.r.p**. may be regarded as a generalization of Robinson's resolution principle. Consider one more example. Let

$$
\begin{array}{ll}
(1)\ D1=\overline{E}(x) \vee V(x) \vee S(x,f(x)) & \\
(2)\ D2=\overline{E}(x) \vee V(x) \vee C(f(x)) & \\
(3)\ D3=P(a) & \qquad\qquad\qquad\qquad \mathbf{(4.2)} \\
(4)\ D4=E(a) & \\
(5)\ D5=\overline{S}(a,y) \vee P(y) & \\
(6)\ D6=\overline{P}(x)\vee\overline{V}(x) & \\
(7)\ D7=\overline{P}(x)\vee\overline{C}(x). &
\end{array}
$$

A covering matrix for these disjuncts is shown in Fig.4.3

	D_1^*	D_2^*	D_3^*	D_4^*	D_5^*	D_6^*	D_7^*	D_8^*	D_9^*	D_{10}^*	D_{11}^*	D_{12}^*	A_{13}^*	A_{14}^*
E				1				1						
$\overline{\mathbf{E}}$	1	1						1						
V	1	1							1				1	1
$\overline{\mathbf{V}}$						1			1					
S	1									1				
$\overline{\mathbf{S}}$					1					1				
P			1		1						1			
$\overline{\mathbf{P}}$						1	1				1		1	
C		1										1		
$\overline{\mathbf{C}}$							1					1		

Fig.4.3

Let us formulate a theorem which establishes a principle (not a method) of logical inference on the basis of **g.r.p**.

Theorem 4.2. Let π be unredundant cover for a covering matrix corresponding to a given system *S* of disjuncts. Then:

(i). If π contains exactly n literals (n is the number of different predicate symbols in S) then S has a model, that is S is not a contradictory system.

(ii). If π contains more than n literals then let us select exactly (n+1) column-disjuncts: $j_1,...,j_{n+1}$ unifiable on the basis of a unificator $\Theta$and find column-resolvent A in the same way as in propositional calculus. Then the disjunct, obtained by application of Θ to A (denoted as A^{Θ} or A_{Θ}), represents logical consequence of disjuncts $j_1,...,j_{n+1}$.

(iii). If A= $\varnothing$ then *S* is contradictory (if A $\neq$ $\varnothing$ then A_{Θ} should be added to *S* in order to provide the following iteration(-s) of **g.r.p**. on the extended covering matrix).

Evidently, the unifiability of disjuncts $j_1,...,j_{n+1}$ directly follows from Herbrand's theorem [7,9]. This requirement, however, may be sometimes unrealizable. Therefore, we should give necessary details showing how to apply **g.r.p**. in these cases. Return to the previous example.

For the matrix in Fig.4.3 we select six column-disjuncts: D2*, D4*, D7*, D9*, D10*, D11* and find their resolvent A13*. Since non-tautological disjuncts D2, D4, D7 are unifiable we obtain a representation of A13* in the form

$$A13 = V(a) \vee \overline{P}(f(a))$$

(on the basis of substitution x=a). Note that here and further only non-tautological disjuncts must be unified. Now select column-disjuncts D1*, D4*, D5*, D9*, D12*, and A13* and find their resolvent A14*. Finally, column-disjuncts D3*, D6*, A14*, D8*, D10*, D12* produce an empty resolvent and are unified by substitution x=a. From this one can conclude that initial system (**4.2**) is contradictory.

Let us prove point (ii) of theorem 4.2. According to conditions, stated in this point, there exists a substitution Θ which unifies column-disjuncts $j_1,...,j_{n+1}$. Therefore,

$j_1,\dots,j_{n+1}$ are some ground examples of disjuncts $D^{\Theta}_{j1},\dots,D^{\Theta}_{j(n+1)}$ to which theorems 4.1, 4.2 become applicable. It is clear that a column-resolvent represents a logical entailment from disjuncts $D^{\Theta}_{j1},\dots,D^{\Theta}_{j(n+1)}$.

Now consider a variant in which selected column-disjuncts are non-unifiable. For instance, consider the following system:

$$\begin{aligned}
&D1 = P(f(x)) \vee R(a,y) \vee Q(y)\\
&D2 = \overline{R}(a,b) \vee P(z) \vee \overline{R}(c,z)\\
&D3 = \overline{P}(v)\\
&D4 = Q(c) \vee \overline{Q}(b) \vee R(c,d)\\
&D5 = \overline{Q}(w).
\end{aligned} \qquad \mathbf{(4.3)}$$

Select four disjuncts (as earlier,the number of disjuncts selected must be equal to n+1 where n is the number of different predicate symbols): D1,...,D4. Consider the substitution:

Θ: $\{y=b, z=v=f(x)\}$

which gives

$$\begin{aligned}
&D1^{\Theta} = P(f(x)) \vee R(a,b) \vee Q(b)\\
&D2^{\Theta} = \overline{R}(a,b) \vee P(f(x)) \vee \overline{R}(c,f(x))\\
&D3^{\Theta} = \overline{P}(f(x))\\
&D4^{\Theta} = Q(c) \vee \overline{Q}(b) \vee R(c,d).
\end{aligned} \qquad \mathbf{(4.3')}$$

Let us make the following assumption (φ):

$Q(c)$ is false

$R(c,d)$ is false

$\overline{R}(c,f(x))$ is false

which gives

$$
\begin{aligned}
&D1^{\Theta} = P(f(x)) \vee R(a,b) \vee Q(b)\\
&D2^{\Theta} = \overline{R}(a,b) \vee P(f(x))\\
&D3^{\Theta} = \overline{P}(f(x)) \qquad\qquad (4.3'')\\
&D4^{\Theta} = \overline{Q}(b).
\end{aligned}
$$

The corresponding fragment of the covering matrix is of the form

	D_1^*	D_2^*	D_3^*	D_4^*
P	1	1		
$\overline{P}$			1	
Q	1			
$\overline{Q}$				1
R	1			
$\overline{R}$		1		

One can conclude that if φ is true than it is possible to produce a resolvent of the system (4.3") in the form of literal P(f(x)). Thus, we have

$$\forall x(\varphi \rightarrow P(f(x))$$

or

$$\forall x(\overline{Q}(c) \& \overline{R}(c,d) \& R(c,f(x)) \rightarrow P(f(x)).$$

Note that in R(c,f(x)) and P(f(x)) the arguments are concerted due to the substitution Θ. Thus, we obtain a resolvent of disjuncts D1,...,D4 in the form

$$Q(c) \vee R(c,d) \vee \overline{R}(c,f(x)) \vee P(f(x)). \qquad (4.4)$$

The general idea is practically evident:

- find some (partly) unifying substitution Θ;
- make a non-contradictory supposition φ excluding some literals such that the remaining literals become unifiable in this substitution Θ ;
- produce a resolvent A_{Θ} ;

- produce a general resolvent of the selected column-disjuncts in the following form

$\varphi \rightarrow A_{\Theta}$.

This general depiction will bę reffered to as an "extended resolution scheme".However, we do not give the details of the extended resolution scheme because there are more than one variant of its realization in general. Nevertheless, the following principles must be observed:

- Θ is selected in such a manner that it should provide participation of each column-disjunct in the group resolution (that is, at least one literal of each disjunct must remain after excluding non-unifiable literals (if there are such literals in substitution Θ);
- φ must be consistent in Θ .

Let us prove the following theoretical result.

Assertion. The group resolution strategy on the basis of an extended resolution scheme is "complete",that is an empty disjunct can be deduced from any contradictory system of disjuncts.

Proof is condubted by the demonstration of the fact that for arbitrary two disjuncts D_α and D_β producing Robinson's resolvent $D^{Rob}_{\alpha,\beta}$ there exists an inference of $D^{Rob}_{\alpha,\beta}$ on the basis of **g.r.p.**.

Besides, the relationship

$D^{Rob}_{\alpha,\beta} \vdash D^{g.r.p.}_{\alpha,\beta}$

is always correct.

Consider two cases:

Case 1. D_α and D_β are unifiable;

Case 2. D_α and D_β are not unifiable.

In case 1 it is convinient to consider illustration in Fig.4.3. Take disjuncts D5 and D7 from (4.2) and find their Robinson's resolvent

$\overline{S}(a,y) \vee \overline{C}(y)$.

According to group resolution strategy one should select six column-disjuncts. Two of them D_5^* and D_7^* correspond to disjuncts D5, D7. The others are selected from the tautological disjuncts: $S \vee \overline{S}$ (provides inclusion of literal $\overline{S}$ in column-resolvent), $C \vee \overline{C}$ (provides inclusion of literal $\overline{C}$ in column-resolvent), etc., excluding, evidently, disjunct $P \vee \overline{P}$ since literal $P(\overline{P})$ is cut. The general idea is quite obvious:

"it is necessary to exclude tautological column-disjunct which correspond to a cut pair of literals".

The correctness of this exclusion is practically evident. From this,one can see that the group resolution strategy makes it possible to generate all the resolvents produced by Robinson's resolution principle in cases similar to *Case 1*.

Consider *Case 2*. Let us start with the following illustration:

$$D_\alpha = R(x,y) \vee \overline{R}(y,x) \vee P(x,z)$$
$$D_\beta = R(a,b) \vee P(w,a).$$

Let us, for instance, use a substitution

$$\Theta = \{ y=a,\ x=w=b,\ z=u \}$$

which gives

$$D_\alpha^\Theta = R(b,a) \vee \overline{R}(a,b) \vee P(b,u)$$
$$D_\beta^\Theta = R(a,b) \vee P(b,u).$$

Make a supposition φ : R(b,a) is false which allows us to find a general resolvent:

$$D_{\alpha\beta} = P(b,u) \vee R(b,a).$$

It is seen that

$$D\alpha, D\beta \vdash D\alpha\beta.$$

One can also see that $D_{\alpha,\beta}^{Rob} \vdash D_{\alpha\beta}$.

Let us formulate this fact as a general statement which represents a final step in proving the assertion. Actually, $D_{\alpha\beta}$ is obtained from D_{α}, D_{β} on the basis of substitution Θ which unifies some (sub)sets of literals in D_{α}, D_{β} containing a cut pair. The literals which were excluded by supposition φ are concerted in Θ. Therefore, one can write

$$D_{\alpha}= P(\ldots) \vee \ldots \vee R(\ldots) \vee \ldots \vee T(\ldots) .$$

D_{α}^{Θ} takes the form of

$$D_{\alpha}^{\Theta} = P(\ldots)_{\Theta} \vee \ldots \vee R(\ldots)_{\Theta} \vee \ldots \vee T(\ldots)_{\Theta} .$$

It is clear that if $T(\ldots)_{\Theta}$ has been excluded then it is included in a general resolvent . Consider the inference

$$D_{\alpha}^{\Theta}, D_{\beta}^{\Theta}, \overline{D}_{\alpha\beta}^{\Theta} \vdash \square .$$

In $\overline{D}_{\alpha\beta}^{\Theta}$ literal T will be represented as $\overline{T}(\ldots)_{\Theta}$, i.e. it will be cut from D_{α}^{Θ}. This is also right with respect to any other literal excluded by supposition φ. Cutting these literals provides of remaining only those literals in D_{α}^{Θ}, D_{β}^{Θ} which produce a group resolvent according to theorem 4.2 (point (ii)).

Hence, we obtain

$$D_{\alpha}^{\Theta}, D_{\beta}^{\Theta}, \overline{D}_{\alpha\beta}^{\Theta} \vdash \square .$$

Substitution Θ used in producing Robinson's resolvent is more general than that one used by **g.r.p**. This ends up the proof of the assertion.

So, we have established that **g.r.p.**, utilizing extended resolution scheme, represents a generalization of Robinson's principle because any inference on the basis of that former can be reproduced by means of **g.r.p.** The question is shifted to practical realizations of **g.r.p.**-based inference strategies. One such a strategy is suggested below. It combines **g.r.p**. and Robinson's resolution strategy in such a way that **g.r.p.** provides coordination and acceleration of an inference. Note that other variants are also possible, for instance, that one based on Herbrand's theorem.

4.4. Implementation of group resolution principle

4.4.1. Preliminary remarks

Let us start with the following

Definition. Disjunct D with one or less occurences of each literal P(...) and one or less occurences of its negation $\overline{P}$(...) is called a simple disjunct.

Example.

$P(x,y) \vee \overline{P}(a,z) \vee \overline{Q}(y).$

Disjunct which is not simple will be called complex.

Example (of a complex disjunct):

$P(x,y) \vee P(a,y) \vee R(a) \vee \overline{P}(a,z) \vee \overline{Q}(y).$

An advantage of simple disjuncts is connected to theorem which establishes conditions of generating a unique column-resolvent on the basis of **g.r.p**. In fact, in this case the following conclusion can be drawn:

(A*) . If resolvent A_Θ is produced without supposition φ then A_Θ is unique (see theorem 4.2)

(B*). Suppose that extended resolution scheme was used. Then if there were no literals in a cover excluded by supposition φ as false then a group resolvent

$\overline{\varphi} \vee A_\Theta$

is unique as well. One should think of a unique resolvent as a disjunct R represented in a covering matrix M by the corresponding unredundant column R .

The conditions **(A*)**, **(B*)** are rather important:

if one of them is satisfied each time when a group resolvent is generated then an empty column-resolvent will be produced for a finite number of group resolutions.

Condition (**B***) is easier then condition (**A***) because (**B***) may be satisfied when finding a cover π for a covering matrix. It should be clear that if column-disjuncts, producing group resolvent, are simple then satisfaction of condition (**B***) means that contrary literals, say $T(...)$ and $\bar{T}(...)$ such that $T(...), \bar{T}(...) \in \pi$ are unifiable.

A more complicated situation is connected with complex disjuncts.

Let us consider the following example.

$D1 = P(f(x)) \vee R(a,y) \vee Q(y)$

$D2 = \bar{R}(a,b) \vee \bar{R}(c,z) \vee P(z)$

$D3 = \bar{P}(v)$

$D4 = Q(c) \vee \bar{Q}(b) \vee R(c,d)$

$D5 = \bar{Q}(w)$,

Let us select for an unredundant cover $\pi = \{ R, \bar{R}, \bar{P}, \bar{Q} \}$ column-disjuncts D1, D2, D3, D5. There are two contrary literals R and $\bar{R}$ in π which must be unified. However, there are two occurences of literal $\bar{R}$ ($\bar{R}(a,b)$, $\bar{R}(c,z)$) in D2 which cannot be unified. Therefore, making a group resolvent of disjuncts D1, D2, D3, D5 provides no warrants for obtaining unique resolvent. The reason is that D2 is not a simple disjunct. We are to point out necessary instructions for this case. Let us proceed from one important technique called disjunct exclusion (operation).

4.4.2. Disjunct exclusion

Definition. P-resolvent of disjuncts Di, Dj is called their Robinson's resolvent with a cut literal P ($\bar{P}$).

Definition. Disjunct exclusion consists of replacement of a given disjunct D containing literal P ($\bar{P}$), by all possible P-resolvents of this disjunct D and the other disjuncts of the system $\mathcal{S}$.

Assertion. Excluding disjunct Di on the basis of the above introduced operation from the system S results in obtaining a new system S' such that S and S' are equivalent in the following sense: If S is a contradictory system then S' is a contradictory system too and vice versa.

Disjunct exclusion operation is valid under the following conditions:

(i) If Di is excluded on the basis of P-resolution then there must be only one occurence of literal P(...) in Di and no occurences of literal $\overline{\mathrm{P}}$ (...) .

(ii) If P-resolvent of this disjunct Di with some other disjunct Dj from S contains literal $\overline{\mathrm{P}}$ (...) unifiable with literal P(...) in Di then this resolvent participates in producing another P-resolvent with Di and so on until P-resolution is possible.

Proof. Let Di be of the following form:

Di= P(t1,t2,...,tz) ∨... ∨ Q(r1,...,rs)

and literal P(...) in Di satisfy condition (i) above. Let us replace Di by all possible P-resolvents of this disjunct Di with the other disjuncts from S allowing P-resolution with Di. It should be clear that if new system S' is contradictory then S is contradictory too. So, we need only to prove that if S' is not a contradictory system then so is S. Assume that an opposite supposition is true, that is, $S'$is not contradictory but S ,on the contraty, is. It should be clear that there exists an interpretation Θ in which S' is true and Di is false. Let

Dj= $\overline{\mathrm{P}}$(q 1,...,qz) ∨... ∨ S(s1,...,sk)

be another disjunct wich has a P-resolvent with disjunct Di and let their P-resolvent be of the form

$$\mathrm{D}_{ij}^{\Lambda} = \{\overline{\mathrm{P}}(\mathrm{q\,1,\ldots,qz})\}_{\Lambda} \& \mathrm{P}(\mathrm{t\,1,\ldots,tz})_{\Lambda}\} \vee \ldots \vee \mathrm{Q}(\mathrm{r\,1,\ldots,rs})_{\Lambda} \vee \ldots \vee \mathrm{S}(\mathrm{s\,1,\ldots,sk})_{\Lambda}$$

where a cut pair of literals is put into the figure brackets; Λ is a unifuing substitution such that Θ is a particular case of Λ. By supposition, Dij is true in Θ and Di is false in

Θ. Consequently, there exists a true literal L (common for Dj and Dij) which is different from $P(t1,...,tz)_{\Theta}$.

If $L \neq \overline{P}(t1,t2,\dots,tz)_{\Theta}$ then let us set
$P(t1,...,tz)_{\Theta}$ ="TRUE".

Then, in this supposition, Dj^{Θ} and D_{ij}^{Θ} remain true formulas. Suppose, however, that another disjunct, say,

$Df= \overline{P}(m1,\dots,mz) \vee \dots \vee T(h1,...,hl)$

becomes false in Θ .Clearly, Di and Df must produce their P-resolvent D_{if}^{Θ} which should be false in Θ. But this is impossible because S'_{Θ} is a satisfiable system by supposition. One can see that these considerations remain valid for any other similar disjunct Df.

If $L = \overline{P}(t1,t2,\dots,tz)_{\Theta}$ then resolvent Dij must participate in producing another P-resolvent with Di until all literals P, unifiable with L, will be excluded. Thus, the above given considerations remain correct.

Inference strategy definition

Divide matrix M into two submatrices: M' and M" with the same row sets and such that all the column-disjuncts in M' are simple and all the column-disjuncts in M" are complex. Our inference strategy depends upon the form of M' and M".

Variant I. (a) M'= ∅ (all tautological disjuncts are not taken into account), or **(b)** M' contains the single non-tautological column-disjunct, or **(c)** there is no any available disjunct in M' which can be used for generating a group resolvent (i.e. each disjunct in M' has participated in all possible group-resolutions as a characteristic disjunct containing one of the cut literals. Remind the reader that a column-disjunct D is called characteristic for a given unredundant cover set π if one and only one row from π covers D (the details can be found in chapter 1 in the section devoted to Minimum-size cover problem)). In cases **(a)**, **(b)**, **(c)** resolvents are found on the basis of Robinson's resolution principle,

- in cases **(a)**,**(b)** Robinson's resolvents are produced until any two simple disjuncts containing a unifiable contrary pair of literals are generated. The next step is a transition to Variant II.

In case **(c)** it is required to generate at minimum one simple disjunct in order to proceed from Variant II.

Clearly, all the generated simple disjuncts are added to matrix M'.

Variant II. Our inference strategy is connected to finding an unexcessive covering set $\pi = \{i1,i2,...iz\}$ for matrix M at each iteration. If z=n then initial system of disjuncts is not contradictory (n is the number of different literals). So, we shall assume z>n. Covering set π is looked for in the following way. Let us choose some simple disjunct Di, for instance,

$Di = R(...) \vee \; ... \vee \overline{T}(...)$

(Di shoud earlier not be excluded by means of disjunct exclusion operation because in our strategy all such column-disjuncts remain in matrix M but do not participate in group resolution).Choose some literal, say, R(...),in Di provided that $\overline{R}(...)$ does not belong to Di. Look for another disjunct Dj containing literal $\overline{R}(...)$ unifiable with literal R(...) from Di. If there is no such a disjunct Dj then exclude Di from M by means of disjunct exclusion operation. In fact, Di remains in matrix M but does not participate in producing new resolvents. When performing disjunct exclusion operation new disjuncts may be generated and added to matrix M' or M" respectively. Choose another disjunct Dl in M' and repeat procedure by analogy. Clearly, choosing Dl has sense when (and only when) there remain at least two non-tautological disjuncts in M'. Suppose, a required disjunct Dj is found. For clearity, let

$Di = R(a,x) \vee T(x) \vee \; Z(y,x)$

$Dj = \overline{R}(a,b) \vee T(f(a)) \vee Q(b).$

As follows from the general idea of our inference strategy, Di and Dj must be characteristic column-disjuncts for the cover π. This means that literals T, Z, and Q from Di, Dj cannot belong to this cover π. Clearly, the required cover π exists if and only if there is no column Dl in M covered by the rows from the set {T,Z,Q} only. Assume that there is no such a column Dl. Then one can find an unredundant cover π with $n^{\pi} > n$ rows (because the contrary pair of literals is included in π). Thus, the condition either **(A*)** or **(B*)** is satisfied which provides producing a "good" column-resolvent. (Note that this is the reason for leaving in matrix M column-disjuncts corresponding to that ones excluded on the basis of disjunct exclusion operation). Repeat the iterations on matrix M'.

Now suppose that there is a column in matrix M' covered by row(-s) from {T,Z,Q}. In this case column-disjuncts Di and Dj cannot be characteristic with respect to cover π containing contrary literals R and $\overline{R}$. Make Robinson's resolvent of Di and Dj and add it to M' if it is a simple disjunct or to M" if not. Repeat the iterations forbidding to use Di and Dj as characteristic column-disjuncts with respect to the cut pair of literals R and $\overline{R}$. (However, Di and Dj may be used as characteristic column-disjuncts with respect to another contrary pair(s)).

The iterations are performed until an empty column-disjunct is generated. This result means inconsistency of the initial system of disjuncts. If a cover with n rows is found then procedure also stops testifying to consistency of the system.

It is not difficult to show that the suggested inference strategy produces an empty column-resolvent for every contradictory system of disjuncts. Really, group resolution provides generation of a finite number of unique column-resolvents. The other resolvents are deduced by means of Robinson's principle.

Thus, a new inference strategy is suggested representing a generalization of Robinson's resolution principle. The main result consists of making a group resolvent for more than two disjuncts at a time which provides an acceleration of logical inference.

4.5. Reduction algorithm with term re-writing

Here we give a new method of logical inference for the system of non-Horn disjuncts in general (disjunct is called Horn disjunct if it contains no more than one positive literal). The suggested method is based on some important property of Robinson's resolution principle used in disjunct exclusion operation outlined in the previous section. This property enables one to perform an equivalent transformation of a system of disjuncts in order to reduce the number of literals (variables). Besides, an additional improvement is made by using term re-writing procedure which sometimes leads to obtaining a solution without making any resolvents.

4.5.1. Formalisms

Disjunct is a disjunction of letters (literals) taken with negation or without it. In order to represent disjuncts we shall use incompatibility relation symbol (#). Consequently, the notation

$$\#(x_1, x_2, \ldots, x_k)$$

is equivalent to

$$\bar{x}_1 \vee \bar{x}_2 \vee \ldots \vee \bar{x}_k$$

(accordingly to the sense of incompatibility relation). Sometimes (when it is not clear from context) we shall designate a disjunct written by means of #-symbol as #-disjunct. As before, the problem of logical inference is in answering the question "Is the given system *S* of disjuncts satisfiable (consistent) or not". To answer this question one needs to show whether an empty disjunct can be deduced from *S* or not.

4.5.2. Case of propositional system

As was said earlier, the term re-writing procedure is an essential part of the suggested method.

Definition. The term re-writing procedure (**t.r.p.**) consists of the replacement of each occurrence of the letter x_i $(\bar{x}_i)$ by $\bar{x}_i$ (x_i). The letter x_i $(\bar{x}_i)$ is considered an input parameter of **t.r.p.**

Definition. #-disjunct is called positive (negative) if it does not contain negative (positive) letters.

Example.

(i) $\#(x_1, x_2, x_3)$ - positive disjunct

(ii) $\#(\bar{x}_2, \bar{x}_4\ )$ - negative disjunct.

(It should be noted here that example (i) in traditional representation

$\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3$

is negative disjunct and (ii) in its own turn - positive).

Lemma 4.1. If there is no positive (negative) #-disjuncts in the system S then S is trivially satisfied with interpretation $I=\{\bar{x}_1, \ldots, \bar{x}_n\ \}$ $(\{x_1, \ldots, x_n\})$.

Definition. A system S of disjuncts is called **t.r.p.**-resistant if it is not trivially satisfiable (resolvable) system and there is no letter x_i $(\bar{x}_i)$ such that **t.r.p.** with x_i $(\bar{x}_i)$ as its input produces trivially satisfiable system.

Example (of **t.r.p**-resistant system).

$\#(x_1, x_2)$

$\#(x_1, \bar{x}_2)$

$\#(\bar{x}_1, x_2)$

$\#(\bar{x}_1, \bar{x}_2)$.

Lemma 4.2. If an initial system S is not contradictory then S can be reduced to a trivially satisfiable system for a finite number of applications of **t.r.p**.

The main problem consists of the question of how to transform **t.r.p.**-resistant system into trivially satisfiable one? The suggested strategy solves this problem in the following way:

"Each time when a **t.r.p**-resistant system ***S*** is obtained an equivalent transformation H is used such that

H:***S*** $\Rightarrow$ ***S'***

and ***S'*** is equivalent to ***S*** in the following sense:

1) If ***S*** is a contradictory system then ***S'*** is contradictory too;

2) If ***S*** is satisfied in some interpretation I then ***S'*** is satisfied in interpretation I' such that I can be restored from I'."

Definition. x-resolvent of disjuncts #(x,F) and #($\overline{\text{x}}$,G) is called disjunct #(F,G) where F,G are some sets of letters.

Theorem 4.3. Any system ***S*** of #-disjuncts containing letter x_i $(\overline{x}_i)$ can be replaced by an equivalent (in the above stated sense) system ***S'*** which consists of all possible x_i-resolvents derivable in ***S*** and those disjuncts from ***S*** which do not contain the letter x_i $(\overline{x}_i)$ (tautological disjuncts should be excluded from ***S'***).

Proof can be obtained in the same way as for disjunct exclusion operation.

Example. Let ***S*** be of the form

#(a,b)
#($\overline{\text{a}}$,c,d)
#($\overline{\text{a}}$, $\overline{\text{b}}$,e)
#(a,e)
#(a,$\overline{\text{b}}$, $\overline{\text{e}}$).

Then, producing a-resolvents, find an equivalent system ***S'***:

#(b,c,d)
#(e,c,d)
#(e,$\overline{\text{b}}$)
#(c,$\overline{\text{b}}$, d, $\overline{\text{e}}$).

Theorem 4.3. is used as a regular basis for **t.r.p.**-realization in case of **t.r.p.**-resistant system. Let us consider one full example.

S=

$\#(a,b,c)$

$\#(\bar{a},b,d)$

$\#(\bar{b},\bar{c})$

$\#(\bar{b},d)$

$\#(a,d,c)$.

Obtain an equivalent system on the basis of **a**-resolvents:

S'=

$\#(b,c,d)$

$\#(\bar{b},d)$

$\#(\bar{b},\bar{c})$.

Then **t.r.p** with parameter "c" gives trivially satisfiable system with solution I'=$\{\bar{b},\bar{c},\bar{d}\}$. Restore a solution I from I' using all the re-writings made by **t.r.p.** and find I=$\{\bar{b},c,\bar{d}\}$.

Let us sum up **t.r.p.** in case of propositional system:

1. Apply **t.r.p**. until **t.r.p.**-resistant system is obtained. (The possible issues of a trivially resolvable system deliver no difficulties.)

2. Find x-resolvents (for some letter x in **t.r.p**.-resistant system) and obtain an equivalent system ***S'***. Proceed from point 1.

4.5.3. Case of predicate calculus

Let us start with the following example:

D1= $\#(P(a),Q(x),\overline{R}(b))$

D2= $\#(\overline{P}(b),\overline{Q}(c))$

D3=$\#(\overline{P}(y),\overline{R}(v))$

$D4=\#(P(a),\overline{Q}(x))$

$D5=\#(R(z),Q(z))$.

Since the system is **t.r.p**-resistant, find Q-resolvents:

$D6=\#(P(a),\overline{P}(b),\overline{R}(b))$

$D7=\#(P(a),\overline{R}(b))$

$D8=\#(\overline{P}(b),R(c))$

$D9=\#(P(a),R(x))$

and add disjunct D3 $=\#(\overline{P}(y),\overline{R}(v))$ to these Q-resolvents in order to form equivalent system *S'*. When finding Q-resolvents we unify predicate arguments as usual. Now, find P-resolvents (there are only two of them):

$D10=\#(\overline{P}(b),\overline{R}(b))$

$D11=\#(\overline{R}(b))$.

From this, one can establish

R(b)="TRUE".

Let, for instance,

P(b)="TRUE".

Using this assignments, one can obtain:

R(b)="TRUE"

$\overline{R}(x)=$"FALSE" ($\forall x\overline{R}(x)=$"FALSE")

P(b)="TRUE"

$\overline{P}(a)=$"TRUE"

$\overline{P}(y)=$"FALSE" ($\forall y\overline{P}(y)=$"FALSE")

Q(z)="FALSE".

We need some general basis for obtaining an interpretation satisfying initial system of disjuncts. This basis consists of a regular application of Robinson's resolution principle. Thus, if, for instance, R(b)="TRUE" and there is disjunct #(P(a), R(x)) then one can obtain (by means of Robinson's resolution principle) disjunct #(P(a)) ($\overline{P}(a)$ ="TRUE"). This basis is sufficient in order to restore an interpretation I for the initial system ***S*** of disjuncts when moving from trivially resolvable system ***S'*** to ***S***.

Theorem 4.4. Every system ***S*** of #-disjuncts can be replaced by their P-resolvents (for some predicate symbol P used in ***S***) if none of them contains predicate symbol P ($\overline{P}$). These resolvents and the other disjuncts from ***S*** not containing symbol P ($\overline{P}$) form a new system ***S'*** which is equivalent to ***S***.

Note that if some P-resolvent contains symbol P ($\overline{P}$) then it participates in P-resolution once again (if it is possible). All tautologies are excluded from ***S'***.

Definition. A system ***S*** of #-disjuncts is called P-contractable (P-contracted) if it can be replaced by an equivalent system ***S'*** which does not contain literal P(...) ($\overline{P}$(...)).

Theorem 4.5. A system ***S*** of #-disjuncts is contradictory if and only if one can deduce two contrary disjuncts from ***S***, e.g. #(P(x1,...,xn)) and #($\overline{P}$(y1,...,yn)) with unifiable arguments.

Consider another example.

D1=#($\overline{P}$(a))
D2=#(D(y),$\overline{L}$(a,y))
D3=#(P(x),Q(y),L(x,y))
D4=#($\overline{D}$(b))
D5=#($\overline{Q}$(b)).

Find P-resolvent:

D6=#(Q(y),L(a,y))

and delete disjuncts D1,D3. Find L-resolvent:

D7=#(Q(y))

and delete disjuncts D2,D6. This results in obtaining the following equivalent system:

D7=#(Q(y))

D5=#($\overline{\mathrm{Q}}$(b)

D4=#($\overline{\mathrm{D}}$(b))

which is contradictory. According to theorem 4.4. this means that the initial system of disjuncts D1,...,D5 is contradictory too.

Let us prove theorem 4.4.

Let S' do not contain predicate symbol P ($\overline{\mathrm{P}}$) and I' be some satisfying interpretation for *S'*:

$$\mathrm{I}'=\begin{cases}\alpha(t_{\alpha1},t_{\alpha2},\dots,t_{\alpha r})\\ \beta(t_{\beta1},t_{\beta2},\dots,t_{\beta q})\\ \dots\dots\dots\dots\dots\dots\dots\dots\\ \gamma(t_{\gamma1},t_{\gamma2},\dots,t_{\gamma s})\end{cases}$$

where α(...), β(...),..., γ(...) are true formulas in I', and t_α, t_β,..., t_γ are their arguments respectively. Then I' can be extended to some interpretation I satisfying initial system *S* of disjuncts. If S is contradictory then S' is contradictory too. Suppose that an oposite is true, i.e. S' is not contradictory but S , on the contrary, is. In this supposition it follows (for some literal f(...)) that

$\mathrm{I}' \rightarrow \mathrm{f}(t_{f1},t_{f2,\dots,}t_{fv}) \in \mathrm{I}$

$\mathrm{I}' \rightarrow \overline{\mathrm{f}}\ (q_{f1},q_{f2},\dots,q_{fv}) \in \mathrm{I}$

(→ is an implication and f(...) and $\overline{\mathrm{f}}$ (...) are unifiable). From this one can conclude that there are two #-disjuncts

#(f($t_{f1},t_{f2,\dots,}t_{fv}$) , Z1)

#($\overline{\mathrm{f}}$ ($q_{f1},q_{f2},\dots,q_{fv}$), Z2)

such that Z1,Z2 both are true formulas in the interpretation Θ unifying two sets of arguments:

$\{t_{f1},t_{f2,\dots,}t_{fv}\}$ and $\{q_{f1},q_{f2},\dots,q_{fv}\}$. But this is impossible because their resolvent #(Z1,Z2)$_\Theta$ is true formula in Θ and Z1, Z2 both are true .

Implementation of logical inference procedure

Let us consider the example:

D1: $\#(\overline{Z}_2)$

D2: $\#(P(x),P(a),Z_1)$

D3: $\#(\overline{P}(y),Z_2)$.

Find P-resolvents:

D4: $\#(P(a),Z_1,Z_2)$ (from D2,D3)

D5: $\#(P(x),Z_1,Z_2)$ (from D2,D3).

One can see that literal P(...) appears in P-resolvents. This means that one can use D4,D5 to produce another P-resolvents (if it is possible) :

D6: $\#(Z_1,Z_2)$ (from D3,D4)

D7: $\#(Z_1,Z_2)$ (from D4,D5).

As a result, a P-contracted system of disjuncts is found. Clearly, it may be impossible (under definite circumstances) to obtain X-contracted system (for definite X). This fact immediately follows from unsolvability of first order logic (predicate calculus). In this case, X-resolution transforms into an endless cycle in which new X-resolvents contain literal $X(\overline{X})$. The following example of disjuncts represents non-contracted system:

$\#(R(x,y),R(y,z),\overline{R}(x,z))$

$\#(R(a,b))$.

From this example one can see a definite advantage of **t.r.p**., because **t.r.p.** recognizes the above-given system as trivially resolvable. Consequently, in some cases one can reckon upon the successful application of **t.r.p** to potentially non-contractable systems of disjuncts.

Our nearest task is to point out one formalized inference procedure based on the introduced theoretical background. This procedure consists of the following:

p.1. In order to produce X-resolvents one should first of all use literal X (X=P) which has no more than one occurence in each disjunct (provided that such a literal P ($\overline{P}$) exists).

p.2. Assume that one has found P-resolvent that contains literal P ($\overline{P}$) . Then this resolvent gets a status of a temporarily excluded disjunct if (and only if) one of the following conditions is observed:

2.1. Total number of literals P ($\overline{P}$) remaining unexcluded in this P-resolvent does not exceed a maximum number of literals P ($\overline{P}$) in one of the disjuncts producing this resolvent.

2.2. There is some disjunct D which participates in the inference of this P-resolvent for the second time provided that the same literal P ($\overline{P}$) in D is cut.

Rules **2.1, 2.2** are oriented to obtaining P-contracted system. These rules are proved later.

p.3. Suppose that we subsequently use literals

$$P_1(\overline{P}_1), P_2(\overline{P}_2), \ldots, P_z(\overline{P}_z)$$

and obtain $P_1-, P_2-, \ldots, P_z-$contracted systems respectively. Denote these systems as $S_1, S_2, \ldots, S_z$ and let S be initial system (it is indexed with index 0). Suppose that one has found that S_z is a contradictory system. Then S is contradictory too as directly follows from theorems 4.2 - 4.4.

p.4. Let system S_z be satisfiable. (In particular, if S_z contains no disjuncts at all then S_z is satisfiable.) It should be clear that S_z is a satisfiable system if it may be contracted without obtaining a contradiction. If no temporarily excluded disjuncts have been generated then one can conclude that S is a satisfiable system. However, it may occur that S_z is a satisfiable system but S, on the contrary, is not. This fact may take place when (and only when) there are temporarily excluded disjuncts among the others. It is a well-known fact that Robinson's resolution strategy is semi-resolvable (i.e. it provides inference of an empty disjunct for any contradictory system S). Therefore, to preserve this property of our inference strategy one needs to restore all the temporarily excluded disjuncts (i.e. to abolish their status of temporarily excluded disjuncts). After that, one should resume an inference from a system S_i (i ∈ 0,1,...,z) in which such disjuncts have appeared for the first time.

Let us prove that the above given rules provide obtaining an empty disjunct if (and only if) initial system S of disjuncts is contradictory. Suppose that some system S_i (say, Sz) is satisfied by interpretation I*. Make a supposition that I* cannot be extended to an interpretation I** satisfying the previous system S_{z-1}. This means that there are some logical entailments from S_{z-1} not included in Sz.

Let us say that disjunct φ cuts an interpretation I* if φ is not satisfied by I* (φ is false in I*).

Now we show that in our suppositions there exists some disjunct φ deducible in S_{z-1} which cuts this interpretation I*. There are two cases which are possible:

Case I. System S_{z-1} is contradictory. But if so, then one can obtain some contradictory P-contracted subset of S_{z-1}.

Case II. System S_{z-1} is satisfiable. Assume that any disjunct λ such that

$\lambda \to \varphi$

cannot be deduced from S_{z-1}. In such a case I* satisfies S_{z-1} which entails a contradiction. Therefore, λ can be deduced from S_{z-1}. Hence, one can conclude that if S_{z-1} is false in I* then it is possible to infer an empty disjunct in this interpretation I*. Consequently, there exists contracting P-resolution providing exclusion of all the literals P ($\overline{P}$) from S_{z-1}. Clearly, this contracting P-resolution produces cutting disjunct λ we are interested in. One can also see that λ is false in I* These considerations make up a proof of the following

Theorem 4.6. If P-contracted system is satisfied by interpretation I* no one extension of which gives an interpretation I satisfying initial system S of disjuncts then there exists disjunct λ deducible on the basis of P-resolution and cutting this interpretation I*.

All that remains is to show that our scheme of creating temporarily excluded disjuncts provides obtaining P-contracted system.

Any resolvent which is obtained from disjuncts D1,D2,...,Dn is called Di-based resolvent (disjunct), i=1,2,...,n .

It should be clear that at least one disjunct from initial system S ={D1,...,Dn} takes part in the inference of disjunct Cm. In an appropriate index system one can conclude that "m" is not higher than "n". This remains correct with respect to any other D_{α}-based disjunct Ck. So, there are only restricted-length inferences of each possible resolvent and therefore the number of this inferences in our strategy is finite.

4.6. Conclusion

New logical inference methods described in this chapter have some positive features:

- group resolution is, in general, an acceleration of Robinson's principle since it provides participation in resolution making more than two disjuncts ; also it enables one to obtain only unique resolvents ;
- reduction algorithm with term re-writing may establish satisfiability of a system of disjuncts (that is, it is not oriented to inference of an empty disjunct only as the most of intherence strategies do).

We outlined a role of theoretical logic in problem solving. This role is dual: on the one hand logic is regarded as one of the technologies of solving problems and on the other hand, logic provides suitable programming paradigm realizible within frames of **CAPSS**. This second aspect is considered in the following chapters alongside with the other aspects of the implementation of programming conception of **CAPSS**.

Chapter 5

PROGRAMMING CONCEPTS IN PROBLEM SOLVING

Abstract. In this chapter we shall consider non-traditional approaches to programming techniques in problem solving. Strictly speaking, we shall consider the programming paradigm to be non-traditional if it does not fall within the boundaries of an imperative programming concept.

Although functional and logic programming could be recognized as non-traditional in the above mentioned sense, they cannot be considered suitable for our aims. One of the evident drawbacks of these paradigms is in the assumption of the closed domain of the problem on the one hand, and a lack of means for manipulating the task concept on the other.

The first ideas for non-traditional paradigms are connected with heuristic programming. A.Newell, J.C.Shaw and H.A.Simon suggested a form of universal solving programs [57] GPS and LT with the basic solution principle based on eliminating differences between the current and the desired states of a problem. However, this principle proved to be insufficient even in proposition calculus. But this particular idea was remarkable for extending traditional programming boundaries. The following attempts to move the solving concept to the theorem proving area are worthy of special mention.

5.1. Programming or theorem proving ?

Wang Hao in [58] put forward the idea of employing programming in the sense of theorem proving, or more precisely, for the mehanization of mathematics. The main advantages of this approach were connected with highly developed mathematical theories as an important premise of its implementation. This idea was supported by the soviet scientific school headed by academician V.M.Glushkow [59]. Consider their concept in more detail. The main idea lies in making human-machine systems to build algorithms oriented to theorem proving. The computer is used as a tool in the hands of the experienced mathematician. With the help of a specialized language the mathematician writes down the text which performs the transformation of one state of the theory to the another. Each state is connected with the definite set S of formulae. The set S is consistent, if there exists a formula f for which ($S \vdash f$) is false. The text consists of the set of sentences, definitions, assertions, assumptions, theorems and lemmas which are also referred to as expressions. Consider the main rules for making expressions.

Schemes of expressions

A primary expression is defined as follows

$$\varphi(x_1,\dots,x_n)=p_1x_1p_2x_2\dots p_nx_np_{n+1} \qquad (5.1.)$$

where $p_1,\dots,p_{n+1}$ - some words in the theory's alphabet, $n\geq 0$.

A function may be defined by the scheme $p_1(\ )\dots p_n\ (\ )p_{n+1}$ with free positions, delimited by round brackets and used for variable-arguments. A finite set of the rules **(5.1)** defines some context-free language in Chomsky's classification. Examples of expressions :

( ) belongs to ( );

( ) does not belong to ( );

function from ( ) to ( );

( ) is ( ).

Schemes of sentences:

let ( )

suppose ( )

since ( ) then ( )

denote ( ) by ( )

Schemes of assertions:

( ) ∈ ( )

( ) = ( )

( ) ≠ ( )

( ) ~ ( )

( ) and ( )

( ),( )

( ), where ( )

( ) is divided by ( )

( ) ⊂ ( )

( ) is simple

( ) is equivalent to ( )

there exist ( ) ,etc.

Schemes of terms:

integer number ( )

natural number ( )

set ( )

ring ( )

field ( )

ideal ( )

element ( )

homomorphism ( ) into ( )

linear space over ( ) ,etc.

Example of theorem writing

Theorem.

Let (F) be (finite field).

The number of elements of (F) is (q).

Then there exists (prime number p)

such that (q) = (p^n) where (n) is (natural number).

Proof.

Let (φ) be (homomorphism of the (ring Z) in (F) such that (φ(1)=1 and Ker (φ)≠ {0})). (Ker(φ)) is (the main ideal of (ring(Z))). (Ker (φ))=(pZ)where ((p)∈(Z) and (p) is (prime number)).

(Z/p·Z) is (a field of (characteristic p)). Each automorphism (of the ring (Z/p·Z)) is (identity).

Let (F') be (an image (Z/p·Z) in (F) with (φ)).

Let (n)=(dim(F)).

Let({w_1,...,w_n}) be (a basis of (F)).

(Each element F) may be represented as

(a_1w_1+...+a_kw_k)

where ((a_1,...,a_n) ∈ (F')).

The number of elements of (F) is (p^n).

End.

The most important functional blocks of the proving system are the following:

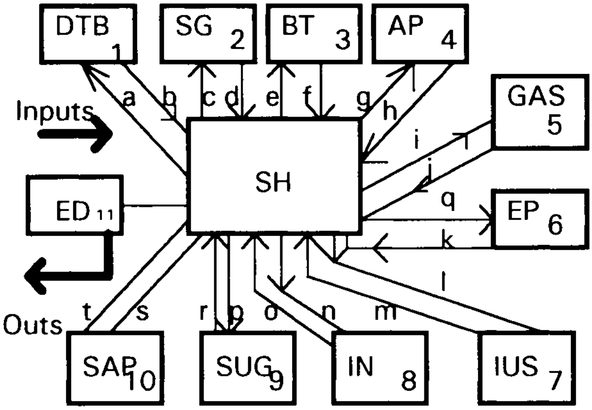
- Definition treatment block (**DTB**)
- assumption processing (**AP**)
- splitting goal (**SG**)

getting auxiliary subgoal (**GAS**)

- block of transformations (**BT**)
- searching unproved goal (**SUG**)
- searching alternative proof (**SAP**)

- scheduler (**SH**)
- equality processing (**EP**)
- induction (**IN**)
- editor (**ED**)
- interface with user (**IUS**)

The blocks above are integrated in the functional structure shown in Fig.5.1.



a:$(S.1)(S_3^4.2)$

b:$(S_1^1.1)(S_1^2.2)$

c:$(S_5^1.1)(S_3^5.1)(S_1^1.1)(S_4^1.1)(S_9^1.1)(S_{10}^1.1)$

d:$(S_2^1.1)(S_2^2.1)(S_2^3.1)(S_2^4.1)(S_2^5.1)(S_2^6.1)$

e:$(S_2^2.1)$

f:$(S_3^1.1)(S_3^2.1)(S_3^3.1)(S_3^4.2)(S_3^5.1)(S_3^6.1)$

g:$(S_1^1.2)(S_2^1.1)$

h:$(S_4^1.1)(S_4^2.1)$

p:$(S_5^3.1)(S_3^3.1)(S_6^1.1)(S_8^1.1)(S_2^5.1)(S_4^2.1)(S_4^9.1)$

r:$(S_9^1.1)(S_9^2.1)(S_9^3.1)(S_9^4.1)$

i:$(S_3^1.1)(S_{10}^3.1)$

j:$(S_5^1.1)(S_5^2.1)(S_5^3.1)$

q:$(S_3^6.1)(S_{10}^1.1)$

k:$(S_6^1.1)(S_6^2.1)$

l:$(S_2^6.1)(S_{10}^4.1)$

m:$(S_1^1.1)$

n:$(S_2^3.1)$

o:$(S_2^1.1)(S_2^2.1)$

s:$(S_9^3.1)(S_2^2.1)(S_2^4.1)(S_5^1.1)(S_3^2.1)(S_6^2.1)$

t:$(S_{10}^1.1)(S_{10}^2.1)(S_{10}^3.1)(S_{10}^4.1)(S_{10}^5.1)$

Fig. 5.1.

The scheduler controls all the rest blocks by means of transferring parameters (S_A^B,R), where S_A^B is a piece of information of block A produced in the operation mode R and B is a piece of additional information.

AP-block has the following inputs and outputs:

S_4^1 - indicates that the last assumption has already been treated;

S_4^2 - points to the contradiction obtained for the new assumption;

GAS-block produces the following information:

S_5^1 - new goal is denied by the checking procedure;

S_5^2 - new goal is obtained;

S_5^3 - an earlier assumption coincides with the subgoal.

EP-block is characterized by the following inputs/outputs:

S_6^1 - equality has been proved;

S_6^2 - there is no sufficient information to prove equality.

IUS-block supplies a user with information S_7.

IN-block deals with the next data:

S_8^1 - a goal can be proved by induction;

S_8^2 - a goal cannot be proved by induction;

SUG-block produces such data items as

S_9^1 - new subgoal has been found;

S_9^2 - all the subgoals have already been proved;

S_9^3 - new subgoal is denied by checking procedure;

S_9^4 - new subgoal is tautologically true;

SAP-block makes next outputs:

S_{10}^1 - new subgoal is an equation;

S_{10}^2 - new subgoal represents transformation of the previously defined formula;

S_{10}^3 - there is no alternative proof of the goal.

ED-block produces an output in the convenient form.

DTB-block is characterized by the following inputs/outputs:

(S_1^1 ,1) - the first subgoal has been treated with all terms defined;

(S_1^1 ,2) - an assumption which has not earlier been countered now is placed into the list of term definitions.

SG-block deals with the next data:

S_2^1 - new subgoal and new assumption are obtained;

S_2^2 - a subgoal is denied;

S_2^3 - a subgoal has the form $\bigwedge_{i=1}^{n} U(i)$;

S_2^4 - a subgoal may not be correctly formulated;

S_2^5 - new subgoal is a tautology;

BT-block can output next information:

S_3^1 - block cannot be applied;

S_3^2 - new subgoal is denied by the checking procedure;

S_3^3 - new subgoal is a tautology;

S_3^4 - new subgoal is not correctly defined;

S_3^5 - terms are not mutually concerted in the "="-equation.

The considered approach raises a number of questions concerning theoretical and practical issues. For example, there is one concerning its universality. However, the whole idea of considering programming as a kind of problem solving in mathematics opens new interesting perspectives. We have pointed out two non-traditional approaches to problem solving programming:

- "universal" algorithm technique and
- theorem proving paradigm.

Nowadays we are witnessing a return to these ideas. Consider them briefly.

The author of [60] directly writes: "A lot of ideas have been suggested to overcome this crisis ... in software engineering and mathematical theory of programs. However, the crisis seems to be getting more and more serious".

One outcome is in "appropriating the technologies cultivated for programming in theorem proving and vice versa". The paper outlines a proof checker system in linear algebra and a proof description language (**PDL**) with the syntax based on Gentzen's natural deduction calculus (NK). **PDL** includes the following objects and rules (we reproduce only a part of them).

5.1.1. Logical connectives

contradiction	⊥ (contradiction)
\	¬ (negation)
&	∧ (and)
|	∨ (or)
→ and ←	→ (if ... then ...)
↔	↔ (... if and only if ...)
all	∀ (for all)
some	∃ (for some)
some!	∃! (there exists only one)
=	= (equality)
:	(... is of type ...)
if A then s else t	

5.1.2. Predicates

≤, <, ≥, >

5.1.3. Types and type constructors

sca	scalar
mat	matrix
nat	natural number
seq(τ)	sequence of τs
perm	permutation
m..n	integer from m to n

5.1.4. Primitive functions

col_size (α:mat):nat	column size of a matrix
row_size (α:mat):nat	row size of a matrix
(a:mat[i:1..col_size(a),	
j:1..row_size(a)]):sca	element of matrix
length (σ:seq(τ)):nat	length of a sequence
(σ:seq(τ)[i:1..length(σ)]):τ	element of a sequence
(seq{i:1..n:nat} f(i):τ):seq(τ)	functional definition
domain (p:perm):nat	domain of a permutation
id (n:nat):perm	identity permutation
inv (p:perm):perm	inverse of a permitation
(p:perm*q:perm):perm	composition
(sum{i:m..n} f(i):sca:sca	finite indexed sum Σ
(prod{i:m..n} f(i):sca):sca	finite indexed product Π

5.1.5. Rules for equality

5.1.5.1. $a = a$ $\qquad\qquad$ $a = a$

5.1.5.2. $$\frac{\begin{array}{c} = \\ \vdots \\ a = b \end{array}}{b \quad a}$$ a=b hence b=a

5.1.5.3. $$\frac{\begin{array}{c} \vdots \\ a = b \end{array} \qquad \begin{array}{c} \vdots \\ b = c \end{array}}{a = c}$$

5.1.5.4. $$\frac{a=b \quad A(a)}{A(b)}$$

5.1.5.5. $$\frac{a=b}{t(a)=t(b)}$$

5.1.5.6. $$\frac{a_1=a_1'}{t(a_1)=t_n(a_n)}$$

5.1.6. Induction

$$\frac{\begin{array}{cc} [n{:}nat] & [A(n)] \\ \vdots & \vdots \\ A(0) & A(n+1) \end{array}}{\forall m{:}nat \quad A(m)}$$

Here and further [A] is equivalent to "assume A";

(⋮) is used to denote logical inference path.

5.1.7. Logical Rules

5.1.7.1. $$\frac{\begin{array}{c}[A] \\ \vdots \\ \bot\end{array}}{\neg A} \qquad \frac{\begin{array}{c}[\neg A] \\ \vdots \\ \bot\end{array}}{A}$$

5.1.7.2. $$\frac{\begin{array}{cc}\vdots & \vdots \\ A & B\end{array}}{A \wedge B}$$

5.1.7.3.

$$\frac{\begin{array}{c}\vdots\\ A\quad B\end{array}}{A[B]}$$

5.1.7.4.

$$\frac{\begin{array}{cc}\vdots & \vdots\\ A\vee B & \overline{B}\end{array}}{A}$$

5.1.7.5.

$$\frac{\begin{array}{ccc} & [A] & [B]\\ \vdots & \vdots & \vdots\\ A\vee B & C & C\end{array}}{C}$$

5.1.7.6 .

$$\frac{\begin{array}{cc}[A] & [\neg A]\\ \vdots & \vdots\\ C & C\end{array}}{C}$$

5.1.7.7.

$$\frac{\begin{array}{c}[A]\\ \vdots\\ B\end{array}}{A\rightarrow B}$$

5.1.7.8.

$$\frac{\begin{array}{cc}\vdots & \vdots\\ A\rightarrow B & A\end{array}}{B}$$

5.1.7.9.

$$\begin{array}{cc} [A] & [B] \\ \vdots & \vdots \\ \\ B & A \\ \hline \multicolumn{2}{c}{A \leftrightarrow B} \end{array}$$

5.1.7.10.

$$\begin{array}{c} [a{:}t] \\ \vdots \\ \\ A(a) \\ \hline \forall x{:}t \quad A(x) \end{array}$$

5.1.7.11.

$$\begin{array}{cc} \vdots & \vdots \\ \\ A(a) & a{:}t \\ \hline \multicolumn{2}{c}{\exists x{:}t \quad A(x)} \end{array}$$

5.1.8. Rules for sum and product

5.1.8.1.

$$\begin{array}{c} [A(n)] \\ \vdots \\ f(n) = g(n) \\ \hline \sum\limits_{A(i)} f(i) = \sum\limits_{A(i)} g(i) \end{array}$$

$$\left[\prod_{A(i)} f(i) = \prod_{A(i)} g(i) \right]$$

5.1.8.2.

$$\begin{array}{c}[A(n)]\\ \vdots\\ \perp\\ \hline \sum_{A(i)} f(i)=0\end{array}$$

$$\left[\prod_{A(i)} f(i)=1\right]$$

Consider the theorem: "Let A be a square matrix. Then Det(A)=(^{t}A)". An appropriate depiction in **PDL** is the following.

```
theory determinant
det (A: square)
:=sum{P:perm <col_size(A)>}
sgn(P)* prod {I:1.. col_size(A)} A[P[I],I]
theorem determinant_of_transpose:
all {A:square} det(A)=det(trans(A))
proof
let A: square be arbitrary
n:=col_size (A)
then n=col_size(trans(A))
det (A)
=sum{P:perm <n>} sgn (P)*prod {I:1...n} A[P[I],I]
by definition
=sum{P:perm <n> sgn (inv (P))*prod {I:1...n} A[inv(P)[I],I]
=sum{P:perm <n> sgn (P)*prod {I:1...n} trans(A)[P[I],I]
since
let P:perm <n> be arbitrary
prod {I:1...n} A[inv(P)[I],I]
=prod {I:1...n} A[inv(P)[I]], P[I]]
=prod {I:1...n} trans (A)[P[I],I]
```

Consider three possible patterns: P1, P2, P3.

The pattern P1 includes both:

- specifying a problem in some logical (formal) language as a set of formulae and then proving goal-formula by means of theorem proving technique;

- compiling non-logical specification and applying formal methods of logical inference to the obtained specification.

One can see that the latter variant is more effective because it provides a possibility of construction of an "external" specification language which can be problem dependent. This feature is essential from the practical viewpoint.

The pattern P2 is oriented to the utilization of a logical proving system as a subsystem of **CAPSS.** In this variant logical proving mechanisms are integrated into intelligent dialog or intelligent helper as the components realizing control and validity testing of the formal transformations made by the user when solving a problem.

Pattern P3 is the most restricted variant of logic utilization in **CAPSS**. It provides answering the questions which requires the answers: "yes" and "no". If the answer could not be found the system gives and indefinite reply. This pattern is helpfull in solving logical problems and will be paid special attention in this chapter.

These three patterns are very important both as an automatized solution technique and as a paradigm of the integrating human- solver in the solution searching process.

It is clear that these patterns should be incorporated in some way into the computer aided problem solving system (**CAPSS**). However, we must emphasize once again that there is something beyond these patterns that the system can achieve.

It is evident that logical formalization of the problem may hardly be overestimated as a formalization tool. Its role in the searching process is rather limited. In particular, a proof represented as a strongly formalized chain of theory sentences, with the goal-statement as a result, does not reflect the human searching activity but mainly represents the product of this activity. We can also see that the user thinks in terms which are not strongly defined and may not even be correct with respect to the objects under his consideration. To find an answer, all the necessary prerequisites should be considered. As we pointed out earlier, the problem considered as a partly defined structu^re is not supplied with all the necessary information. This imposes evident constraints on the use of logic as a universal solving approach.

5.2. Universal algorithm paradigm

We consider three different approaches to the universal algorithm concept. The first one is outlined in [61]. As the authors assert "... there should exist something, call it a universal weak method (**UWM**), that responds directly to a situation by behaving according to the weak method appropriate to the knowledge the agent has of the task. Each weak method can then be characterized by the small amount of knowledge it has about the task. The **UWM** is what is left after this characteristic knowledge is removed". Thus, even though the **UWM** does specify a nontrivial behaviour, it is a simple specification that provides just enough control to search a problem space. Fig. 5.2 may serve as a pattern of **UWM** specification.

The scheme shown in Fig. 5.2. is extremely general and may be regarded as a kind of "try-and-test" -pattern. For very large state-spaces it is unacceptable. However, the programming paradigm based on this, is interesting if we consider additional possibilities.

(i) the possibility of (self)-learning;

(ii) incorporating the heuristics to contract a search area;

(iii) including the universal method as a function of the **CAPSS** external language.

Point (iii) will be paid necessary attention in discussing the language conception for problem solving. Another approach was proposed in [62] by generalizing problem solving techniques such as divide-and-conquer, dynamic programming, tree and graph searching, integer programming, branch-and-bound method ,etc. The principle idea is to solve a parent problem by combining the solutions to various child problems. Thus, the solution to a parent problem p_i is associated with the solution to a set of child problems C_i. If $C_i=\varnothing$, p_i is called a leaf problem, i.e. it may be resolved without reference to other solutions.

Elaboration:

Goal: If the current problem space fails, the goal is unacceptable.

Problem Space: If the current state fails, the problem space is unacceptable.

State: If the current operator fails, the state is unacceptable.

Decision:

Goal: If there is an acceptable available goal, vote for it.

Goal: If there is an unacceptable available goal, veto it.

Problem Space: If an acceptable problem space is associated with the current goal, vote for it.

Problem Space: If an unacceptable problem space is associated to the current goal, veto it.

State: If an acceptable state is in the current problem space, vote for it.

State: If an unacceptable state is in the current problem space, veto it.

Operator: If an acceptable operator is associated with the current problem space, vote for it.

Operator: If an operator has already been applied to the current state, veto it.

Fig. 5.2.

A subset $A_i \subseteq C_i$ is a satisfying subset for p_i if it is sufficient to enable the computation of the solution of p_i. We assume that the solution S_i to a problem p_i depends upon the given satisfying set A_i. Thus, there is a function f which constructs a solution S_i from A_i, i.e. $S_i = f(A_i)$.

Denote by P' a set of (sub)problems with known solutions. This is a dynamic set whose cardinality increases during the execution of the solving procedure. Also, introduce a set $C_i' = C_i \cap P'$. The execution of the algorithm causes the suspension of the problem p_i and maintains that suspension if $p_i \notin P'$ and C_i' is not a satisfying subset for p_i. As soon as this condition does not hold the problem p_i is restored.

The main body of the general solving algorithm may now be presented as follows [62]:

```
{main body}
while (a main problem $p_0 \in P'$)
do (select an unresolved problem with
greatest precedence)
process (selected problem)
od
where
process($p_i$)≡
if satisfying set for $p_i$ is defined
then
$S_i := f_i (C_i')$
$P' := P' \cup \{p_i\}$
restore problems depending on $p_i$ only
else
expand $p_i$
suspend $p_i$
fi
```

The main features of this general scheme are similar to that presented above. From our viewpoint, this scheme is more interesting for the aims of solving combinatorial tasks. We may conclude, however, that the universal algorithm cannot pretend to be a comprehensive framework for a programming paradigm in problem solving. Instead, we should consider the incorporation of weak-method techniques into the searching process that are connected to the definition of their place and means of realization. One of the interesting realizations of the universal paradigm is the system **ALICE** (A Language for Intelligent Combinatorial Exploration) [63].

The development of **ALICE** has been aimed at separating the goal specification of the problem from its solution. It employs a kind of declarative language for specifying problems. The system is oriented towards the class of complicated problems which require a large amount of computer resource. This general class of problems is defined by the common formulation of the form:

"To find unknown X ∈D such that X satisfies the given constraints K(x)".

The set D is supposed to be finite. The alphabet of **ALICE** consists of the next items:

Word	**Meaning**
Let	Definition
TO FIND	Goal
WITH	Restriction
END	End of definition
CST	Constant
ENS	Set
COE	vector of numeric coefficients
COA	vector of alphabetic and numeric coefficients
MAT	matrix
FON	function
ING	injection
SUJ	surjection
BIJ	bijection
SUC	successor
PRE	predecessor
SYM	symmetry
VAL	value
CHE	path
ARB	tree
CIR	scheme
MIN,MAX	min,max

<, >, ≤, ≥, =, ≠, +, -, *, /	
MOD	module
OBJETS	definition of the set through its elements
PDT	set product
INC	including
APP	to belong to ...
→	function definition
QQS	whatever it would be ...
EXI	there exists ...
NON	negation
UN	set union
IN	set intersection
OR, AND	disjunction, conjection

Syntax of **ALICE** is defined by the following.

<formulation>::=<phrase>*<data>

<phrase>::=<description>|<restriction>|END

<description>::=LET CST<constant name>|

LET<vector><vector name> SUR<name>|

LET MAT<matrix name> PDT<name><name>

TO FIND <function><function name>:<name>

→ <name><option>*

<set definition>::= [<constant name>,<constant name>]|

OBJET<object name>*|

UN<name><name>|

IN<name><name>

<vector>::=COE|COA

<function>::=FON|INJ|SUR|BIJ

<option>::=SUC|DIS|DMI|DMA|CIR|ARB|SUM|VAL

<restriction>::=WITH<logical expression>

<logical expression>::=<binary operator><logical expression>

<logical expression>|

NON<logical expression>

<algebraic operator><expression expression>|

MIN<expression>|

MAX<expression>

|<quantifier> APP<name><name>

<logical expression>

<binary operator>::=OR|AND|OU|ET

<quantifier>::=QQS|EXT

<algebraic operator>::=< |> |≥ |≤ |≠

<expression>::=<operator><expression><expression>|

<number>|<name>

<operation>::=+| –|* |/ |MOD

<number>::=<digit>|<number>

<name>::=<letter>|<name>.

Consider an example of the task specification. Let it be necessary to find a Boolean vector x satisfying the following equations:

$$0 \le x_1 x_4 + 3x_2 x_5 - x_1 - 1$$

$$0 \le 2x_1 - 3x_5 + x_4 - 2$$

$$0 \le 2x_2 + x_3 - 3.$$

Specification in **ALICE** has the following form:

LET

SET I=[1,5]

SET B=[0,1]

TO FIND FUNCTION x:I → B

WITH

$0 \le x(1) * x(4) + 3 * x(2) * x(5) - x(1) - 1$

$0 \le 2 * x(1) - 3 * x(5) + x(4) - 2$

$0 \le 2 * x(2) + x(3) - 3$

END

Consider another example. It is necessary to assign each job from a given set T to a processor to minimize the total number of processors engaged and satisfy the criterion below

$$\sum_{t \in T} W(t) \le L$$
$$p(t) = r$$

where W(t) is a processing time of the job t∈T;

p(t) is a processor number which executes problem t.

It is also assumed that all the processors are identical.

This problem may be specified in **ALICE** as follows:

LET

CONSTANTS N,L

SET T={1,N}

COEFFICIENT

W ON T

TO FIND FUNCTION p:T → T

WITH

$\forall r, r \in T$

$$\sum_{t \in T} p(t) \le L$$
$$p(t) = r$$

WITH

$$\underset{\text{p}}{\text{MIN}} \quad \underset{t \in T}{\text{MAX}}\, p(t)$$

END

The main part of the solving algorithm used in **ALICE** is connected with constraint processing. There is a set of strict procedures processing the definite constraint types. A simple try-and-test technique is used if other effective methods cannot be applied. When indeterministic choice is to be employed **ALICE** uses a number of heuristic estimations, such as those shown below

1)the number of constraints containing a given variable;

2)complexity of expressions ;

3)the domain sizes of the variable ;

4)coefficients values ,etc.

A scheme of the constraints analysis is applied to all the constraints in the form

$$T_p - T_n \geq 0 \quad or \quad T_p - T_n = 0.$$

where T_p denotes all variables prefixed by the sign "+", and T_n denotes all members prefixed by the sign "-", for example

in $2x_1 + x_2 - x_3 - 2 \geq 0$

$$T_p = \{x_1, x_2\} \quad and \quad T_n = \{x_3\}.$$

Thus, from the restrictions

$$2x_1 + x_2 - 9 \geq 0$$

$$x_1 \in [2,5], \quad x_2 \in [4,5]$$

ALICE produces new restriction

$$x_1 \geq 2$$

and from the system

$$\begin{cases} x + y = 2 \\ x + 2y + 3z \geq 3 \end{cases}$$

it produces y + 3z ≥ 1 (using substitution x = 2 - y). The reader may find the necessary details in [63].

5.3. Computer mathematics

There are other systems in computer-aided mathematics with the programming paradigm which are remarkable for their incorporation of the human-solver in the solution process as the main component and algorithm maker. We have in mind the computerized algebraic systems, such as, for example, MACSYMA, REDUCE, muMATH, SCRATCHPAD and others. All of these have the following common features [64]:

- the programming process is highly interactive: the user has the opportunity to control and intervene in the computing process at any time;

- almost all the data used are the kind of mathematical expressions written in an ALGOL-like language;

- the majority of the algebraic computing systems used in practice are realized in Lisp.

In comparison with the conceptions considered earlier, this programming paradigm is associated with the scheme which is highly attractive for problem solving. Remember that the principle dividing responsibilities between the human-solver and the computer prescribes that the formal part of the problem be left to the computer and informal part to the human investigator.

However, these systems of computer mathematics are not recognized as problem solving ones because they do not deal with the problem logical structure (regularities) and the search for a solution as it is understood, for example, in heuristic programming.

5.4. Expert systems

The traditional problem solving technology is based on running the program compiled by the human-solver on the computer. The majority of programs consist of an imperative set of precise instructions for the computer. Each instruction is exact and unambigious. We can see that in this case, the program represents a complete version of an algorithm written in the input language of the computer. Two basic points should be clarified when building a programming paradigm of problem solving:

A: *"How to solve a problem (in general)?"*

B: *"How to organize solution process in the human-machine media ?"*

The traditional paradigm is the following:

A: *As it may be solved by every individual.*

B: *Computer simply scans and performs presented solving procedure.*

The expert-system paradigm is as follows

A: *As it may be solved by well qualified experts.*

B: *Computer derives the solution from the knowledge base by means of an inference mechanism.*

As the reader can see, the most significant difference in the latter paradigm is in point "**B**". This is due to the fact that a solving algorithm is considered to be a priori unknown and is searched for on the basis of a logical inference machine or a Bayes' resolution strategy. Thus, the development of informal solving strategies is an important part of the theory of expert systems. The paradigms given above may be represented in other forms. Thus, a traditional conception has the form:

"Data + Algorithm = Program"

similarly, an expert-system paradigm takes the following form

"Knowledge + Inference Strategies = Problem Solving".

In fact, both knowledge and inference mechanisms are tightly connected to each other. For example, Prolog as a knowledge representation language and as a logical inference system, is characterized by the following conceptual scheme

A: *Try-and-test strategy as a universal solving approach*

B: *Solution is derived by computer from the Horn-clauses specification of a problem prepared by human.*

Comparing the above paradigms one can conclude that the expert-systems-based conception is more general; its effectiveness depends crucially upon the knowledge treating mechanisms and the sufficiency to produce desired results.

We should, however, expand the expert system paradigm as shown below to reflect human participation in the solving process:

"Knowledge + Inference strategies + User interface = Problem-solving-system".

When analysing this scheme one should bear in mind the following useful features of the expert systems:

- availability of a knowledge base about the task and its solutions;
- searching for a solution on the basis of inference strategies
- usage of a form of the knowledge representation language

Expert systems mechanisms may be useful in:

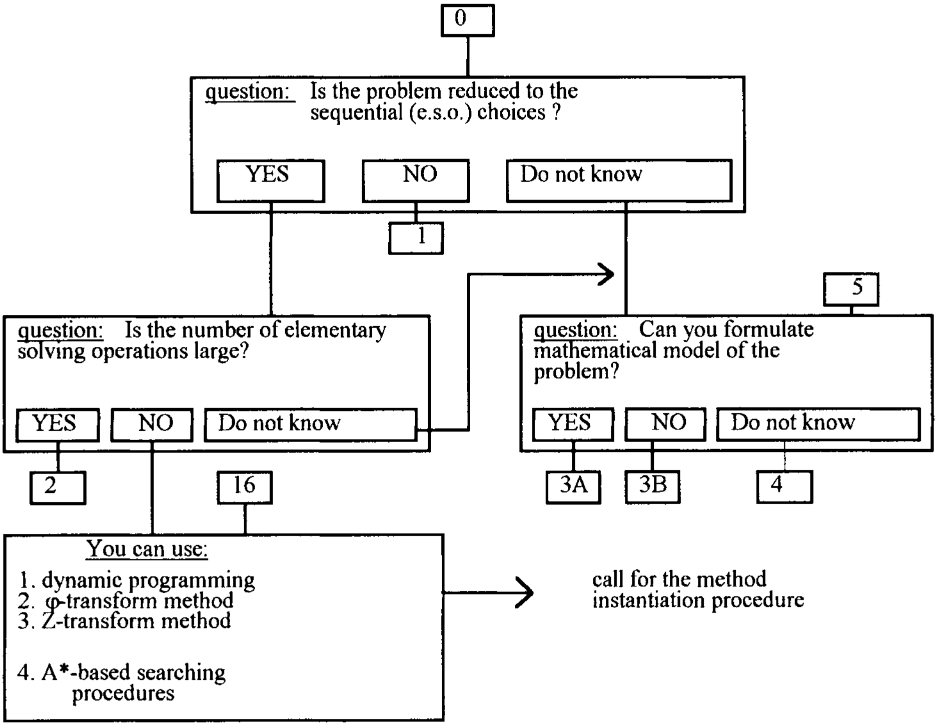
(i) recognizing task types;

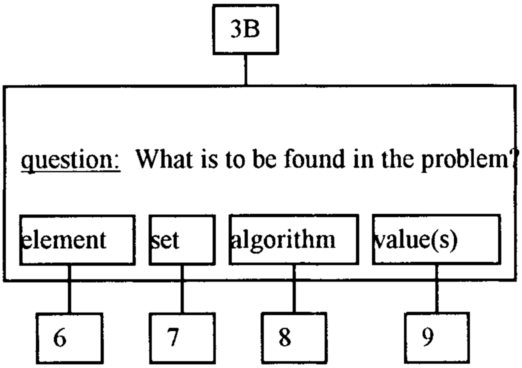
(ii) definition of the applicable solution strategies and heuristics;

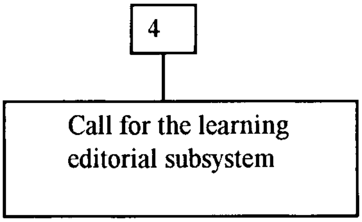
(iii) appreciation of the quality of the solution obtained;

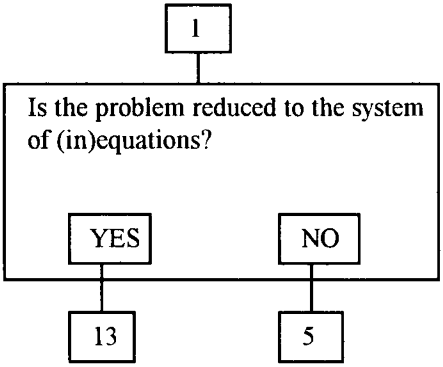
(iv) formulating new subgoals.

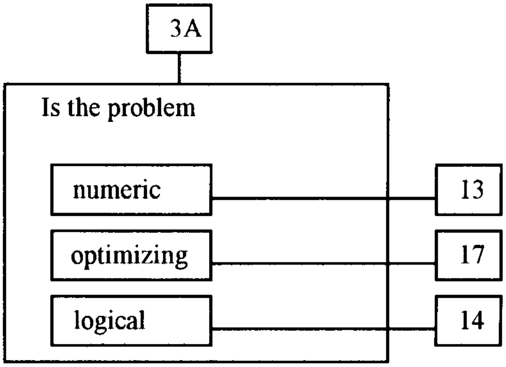
Consider the following frame of consulting procedure used in CAPSS.

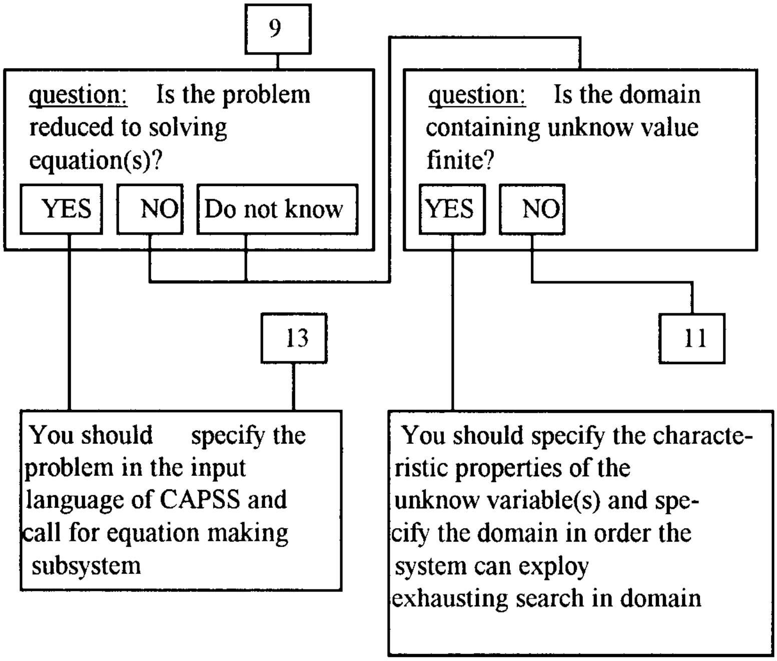

0
question: Is the problem reduced to the sequential (e.s.o.) choices ?
YES
NO
Do not know
1
5
question: Is the number of elementary solving operations large?
YES
NO
Do not know
2
16
question: Can you formulate mathematical model of the problem?
YES
NO
Do not know
3A
3B
4
You can use:
1. dynamic programming
2. φ-transform method
3. Z-transform method
4. A*-based searching procedures
call for the method instantiation procedure

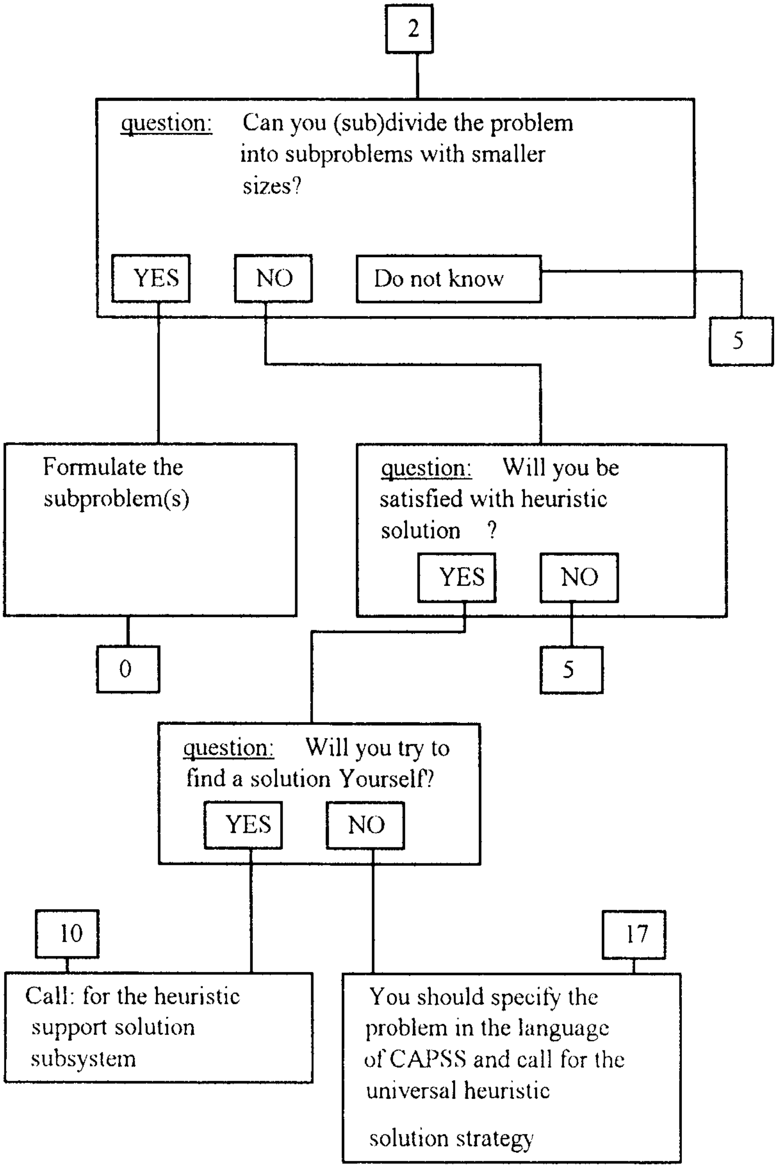

3B
question: What is to be found in the problem?
element
set
algorithm
value(s)
6
7
8
9

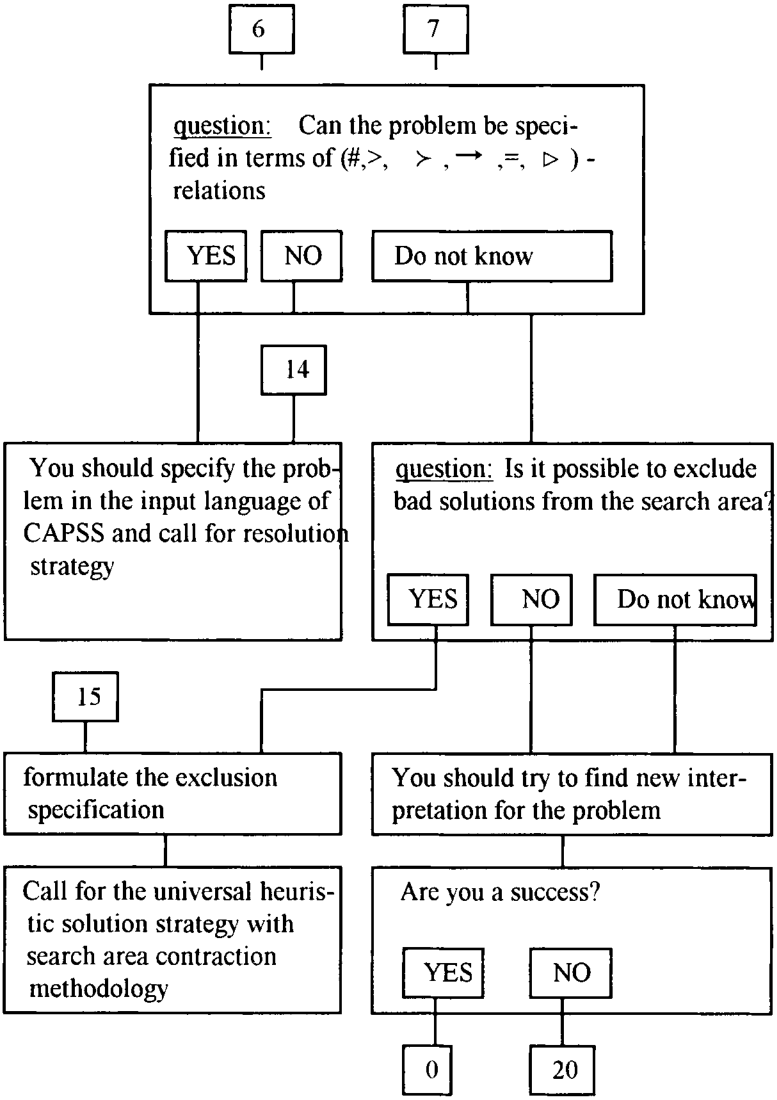

4
Call for the learning editorial subsystem

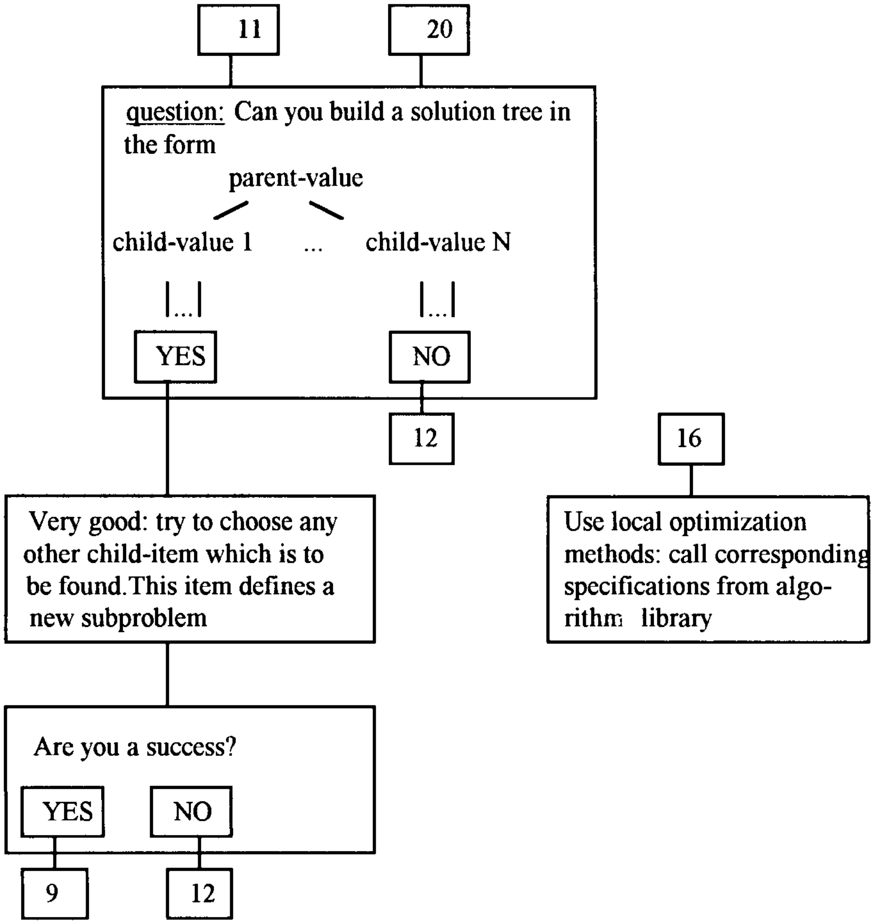

1
Is the problem reduced to the system of (in)equations?
YES
NO
13
5

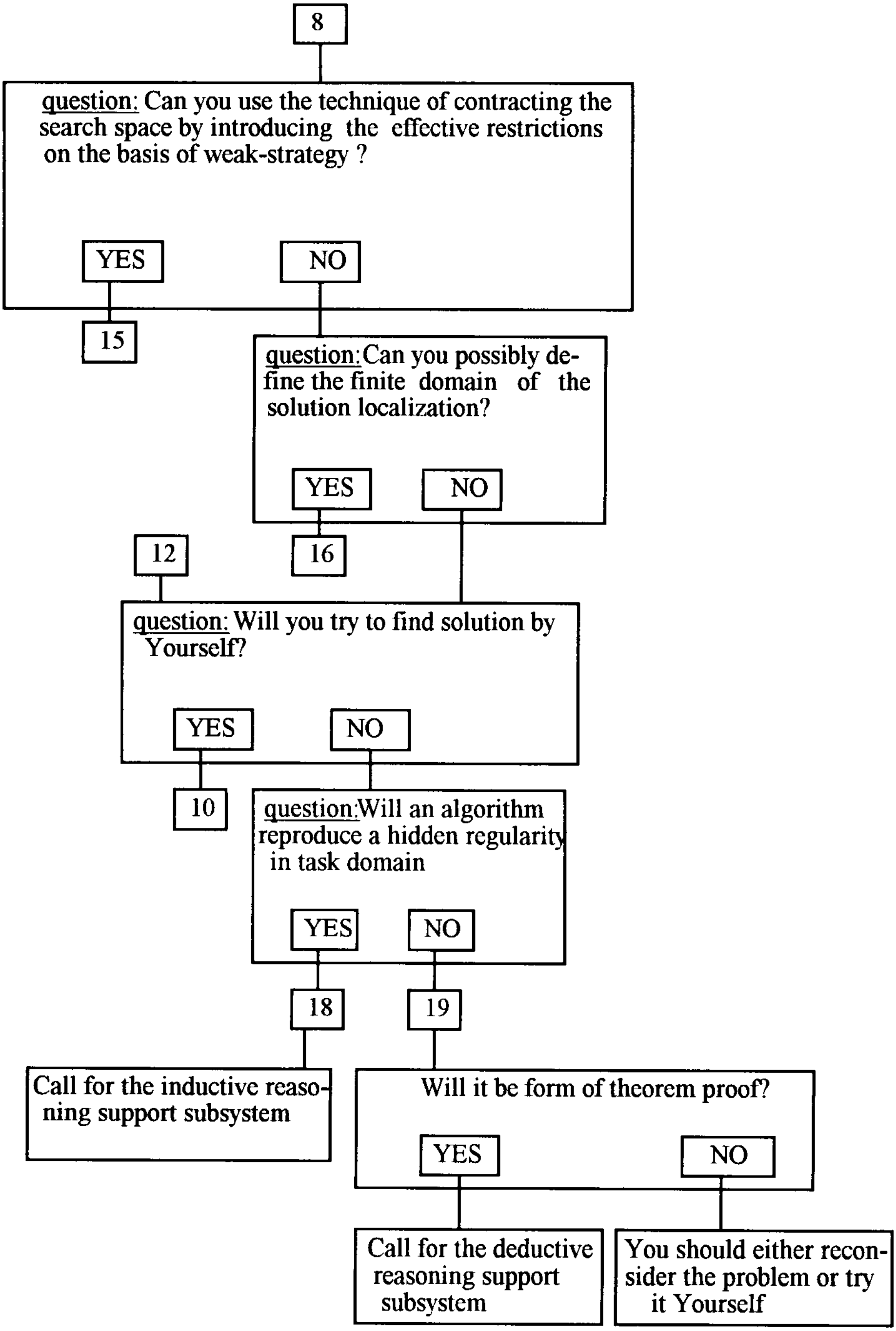

3A
Is the problem
numeric
optimizing
logical
13
17
14


9
question: Is the problem reduced to solving equation(s)?
YES
NO
Do not know
question: Is the domain containing unknow value finite?
YES
NO
13
11
You should specify the problem in the input language of CAPSS and call for equation making subsystem
You should specify the characteristic properties of the unknow variable(s) and specify the domain in order the system can exploy exhausting search in domain


2
question: Can you (sub)divide the problem into subproblems with smaller sizes?
YES
NO
Do not know
5
Formulate the subproblem(s)
question: Will you be satisfied with heuristic solution ?
YES
NO
0
5
question: Will you try to find a solution Yourself?
YES
NO
10
17
Call: for the heuristic support solution subsystem
You should specify the problem in the language of CAPSS and call for the universal heuristic solution strategy


6
7
question: Can the problem be specified in terms of (#,>, ≻ ,→ ,=, ⊳) - relations
YES
NO
Do not know
14
You should specify the problem in the input language of CAPSS and call for resolution strategy
question: Is it possible to exclude bad solutions from the search area?
YES
NO
Do not know
15
formulate the exclusion specification
You should try to find new interpretation for the problem
Call for the universal heuristic solution strategy with search area contraction methodology
Are you a success?
YES
NO
0
20


11
20
question: Can you build a solution tree in the form
parent-value
child-value 1 ... child-value N
|...|
|...|
YES
NO
12
16
Very good: try to choose any other child-item which is to be found.This item defines a new subproblem
Use local optimization methods: call correspondin specifications from algorithm library
Are you a success?
YES
NO
9
12


8
question: Can you use the technique of contracting the search space by introducing the effective restrictions on the basis of weak-strategy ?
YES
NO
15
question: Can you possibly define the finite domain of the solution localization?
YES
NO
12
16
question: Will you try to find solution by Yourself?
YES
NO
10
question: Will an algorithm reproduce a hidden regularity in task domain
YES
NO
18
19
Call for the inductive reasoning support subsystem
Will it be form of theorem proof?
YES
NO
Call for the deductive reasoning support subsystem
You should either reconsider the problem or try it Yourself

5.5. Evolutionary problem solution synthesis (EPSS) concept

The **EPSS** - approach is proposed in [65]. It takes into account the main features of the human mind as an ill-defined system, and difficulties in control mind processes. The model of **EPSS** evolves spontaneously along with the model of the problem solution. It is defined as 4-tuple

$M(EPSS) = < Z,C,D,R_I>$,

where

Z is a language of knowledge objects or knowledge system representation;

C is a calculus of knowledge objects and knowledge system evolution;

D is a model of a problem solution synthesis, and

R_I represents the controller of information interaction between the mind and the external models of the knowledge systems.

The syntax of a language is user-dependant . The process of human-controlled solution synthesis may be interpreted as the growth of a solution tree by means of the following manipulations:

- expansion of the tree nodes;
- returning to previously expanded nodes;
- deleting nodes from the solution tree.

In order to realize these manipulations,the following operations are introduced : generation (Gen(.)) and local synthesis (D(.)) as the elements of calculus C. Each operation transforms the current state $(E_j,M_j,Cont(j))$ to the $(E_{j+1},M_{j+1}$, $Cont(j+1))$ with a new set of the knowledge products T_{j+1}. Here: E_j represents the problem environment; M_j is a solution model and Cont(j) is the structure of formation operations. Each formation operation is an indeterministic rule of the form:

<<Observational description of the required changes in the knowledge objects at E_j and M_j >, <Conditions which have to be satisfied>> → <Operational description of changes of knowledge objects at E_j and M_j >.

Every rational operation causes a decrease of the system entropy. To evaluate this decrease numerically, the notion of Formation Energy F is introduced. The Formation Energy F measures the external representations of the knowledge evolution process with the real function μ: $M_j \rightarrow \{0,1\}$. The law describing the changing of F is represented in the form

$$F_{j+1} = F_j + \alpha(j) \cdot \Delta A_j,$$

where $\alpha(j) = \dfrac{dF_j}{d\gamma_j}$;

ΔA_j - is a subjective measure of the "distance" to the solution emergence, (related to the time j);

γ_j - is the inner formation energy needed by the mind to promote a solving process at the state j.

The inner formation energy γ is hard to be measured directly and it is proposed to employ the dependancy F = F(γ) shown in Fig.5.3. Here, the interval [0,F^L] is the domain of **EPSS** destruction, the interval [F^L, F^N] is the domain of uncertainty, and the interval [F^N,1] is the domain of a succesful run of EPSS.

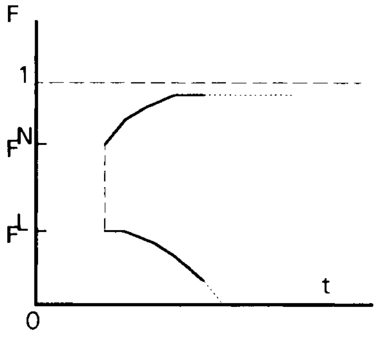


Fig.5.3

The approach briefly outlined above is remarkable for the incorporation of the informal system such as a human mind in the solving process. There are some aspects which should be addressed :

(i) The language L for problem conceptualization is user-defined and not predetermined.

(ii) A form of the problem measuring is designated as Formation Energy.

(iii) The solution Tree is augmented by a strong interaction with user, by means of controller R_I.

5.6. Mathematical induction and pattern recognition approaches

These patterns are aimed at the investigation of semiformalized systems and the revealing of new regularities in the pattern domains. Consider the three series of numbers A, B, C below

A: 4, 3, -6, 5 , 2.5 ...

B: 1, 4, 2, 2.5, -2 ...

C: 9, 7.75, -7, 9.5, -0.75 ...

The pattern recognition program in arithmetic [66] will establish that C = =A/B+A+B. Consider the basic ideas briefly. To derive an arithmetic law for the given number series, the program first creates a set of logical characteristics. Each characteristic is built as follows: a simple arithmetic function $N_i = f_i(A,B,C)$ is calculated (for example, $f_i \triangleq A + B$ or $f_i \triangleq C{:}B$,etc) and then each of the next logical operators is applied to the calculated value N_i :

$$\begin{cases} L_1 N_i = 1, & \text{if } N_i \text{ is an integer} \\ L_1 N_i = 0, & \text{otherwise} \end{cases}$$

$$\begin{cases} L_2 N_i = 1, & \text{if N} \geq 0 \\ L_2 N_i = 0, & \text{otherwise} \end{cases}$$

$$\begin{cases} L_3 N_i = 1, & \text{if } |\mathrm{N}| \geq 1 \\ L_3 N_i = 0, & \text{otherwise.} \end{cases}$$

Given a set of logical characteristics $l_{ij} = L_i N_j$, the program then finds the values of the logical functions $F_m(l_{ij}, l_{kn})$, where F_m is one of the given set of boolean functions:

$$ab, \bar{a}b, a\bar{b}, \overline{ab}, a \vee b, \bar{a} \vee b, a \vee \bar{b}, \bar{a}b \vee a\bar{b}.$$

Then, to generalize the F_m - values to the entire series A, B, C, the following conjunction is used:

$$\sigma_m = F_m[L_{j1} f_1(A_1, B_1, C_1); L_{j2} f_2(A_1, B_1, C_1)] \wedge$$
$$F_m[L_{j1} f_1(A_2, B_2, C_2); L_{j2} f_2(A_2, B_2, C_2)] \wedge$$
$$\wedge \ldots \wedge$$
$$F_m[L_{j1} f_1(A_k, B_k, C_k); L_{j2} f_2(A_k, B_k, C_k)].$$

Not all the characteristics are used though, only those that provide a distinction between different number series and are the same for two different examples of the series with the same arithmetic law.

The above outlined solving pattern is evidently a kind of universal "try-and-test" strategy. There are, however, important issues which directly concern this technique, namely:

- recognizing the problem model within the given classification of models, and
- choosing a heuristic suitable for the given problem specification.

Note, that the pattern recognition technique is also used in inductive logic models [67] and expert system inference mechanisms.

We can also consider inductive reasoning as an attempt to remove the main restrictions of the deductive systems connected with the "closed world principle" assumption. Inductive reasoning is based on the Mill's formal schemes outlined briefly in the previous chapter. Formally, an inductive reasoning system tries to establish two binary relations:

- "object X has properties Y";
- "W is a consequence of a reason V".

An inductive solver uses a set of elementary solving procedures for:

- evaluating the predicates which assert object similarity or dissimilarity. The former are used to find the reasons sufficient for the given phenomenom to arise; the latter are used to find a set of consequences of the given reason;
- checking up the additional conditions used to partition a given set of hypothesis which use the plausibility levels;
- generating the set of consequences from the given predicates instantiated by the number of examples and counterexamples;
- generalizing a number of hypotheses with the aim to establish logical laws which cause different combinations of phenomena;
- establishing new facts which are unknown to user;
- explaining of the results obtained by the induction and verifying their consistency.

These procedures are formalized as the axioms and inference rules, so the whole approach is the same as in the theorem proving systems case.

5.7. Intellectual support concept in the problem solving system

Practical realization of the **CAPSS** has a number of difficulties dependent on:

- insufficiency of pure logic methods, theorem proving, and other techniques for solving various problems;
- impossibility to reproduce effectively the mechanisms of intuition and conjecture;
- difficulties in making interpretations solely on a formal basis ;
- lack of a good theory explaning human measurement of a task as a psychological phenomenon et.al.

The above outlined and other approaches are mainly directed towards automatic problem solution which is also connected with three serious drawbacks:

1. Automatized solving systems inherit an incompleteness property as do all formal systems, including arithmetic, due to the well-known Gödel's incompleteness theorem;

2. In fact, the problem formalized in such systems is specified by means of declarative knowledge sufficient to derive a solution;

3. A user is not directly involved in solution synthesis and loses inerest in finding a solving algorithm.

As we can see these reasons are rather serious for putting forward another programming paradigm different from the "full automatization concept". This paradigm is connected with the supposition that there is no program in the traditional sense. Human - solver has at his disposal a special language (or a number of languages) which enables him to describe a solving process as a set of transformations of the problem's states. The entire process is partly organized under a user's control (to provide intuitive and heuristic behaviour) and partly automatically (in traditional sense). The paradigm includes the following basic elements:

- insertion points and computation patterns
- intelligent oracle
- a problem-free user defined language (**L0**)
- a formal language for problem specification (**L1**)
- an operational user language to manipulate task concepts (**L2**)
- making equation subsystem (**MES**)
- a library of weak wethods (**WML**)
- universal solving strategy and heuristic utilization.

Let us get started to consider these elements in detail.

5.7.1. Languages

To represent different optimization problems, a kind of mathematical language is needed. More exactly, we need some languages.

The following is the syntax of the language L1.

Terms:

(i) constants

(vi) TRUE (true)

(ii) variables

(iii) functors

(iv) ∞ (infinity)

(v) ∅ (empty set)

(vii) FALSE (false)

(viii) NULL (null element)

(ix) _ (anonymous variable)

(x) set

Primitive relations

(i) ¬ (negation)

(ii) & (conjunction)

(iii) ∨ (disjunction)

(iv) → (implication)

(v) ⊸ (prohibition)

(vi) ≻ (precedence)

(vii) || (parallelism)

(viii) : (alternativeness)

(ix) =,≥,> (equality|equal or more|more)

(x) # (incompatibility)

(xi) ∈ (to belong to)

(xii) ∉ (to be a subset of)

Quanifiers:

*	all
!	one of ... (each one of)
* ...\{p}	for all subjects excluding those with the property p
(t)	for the given t subjects ...
∃	there exists ...
?	is to be found
where	such that...

Primitive functions:

(i) +,-,*,/ (addition, subtraction, multiplication, division)

(ii) Σ (i where i $\in$D) ($\sum_{i \in D} i$)

Π (i where i $\in$D) ($\prod_{i \in D} i$)

(iii) max(f where f $\in$D)

min(f where f $\in$D)

(iv) !φ the value of an arbitrary element belonging to φ

(v) *φ the cardinality of φ

(vi)*!φ the sum of all elements in φ. It is equal to Σ(x where x $\in \varphi$)

(vii) value (Z where P(Z)) the value of element Z with property P(Z)

(viii) set (T where P(T)) the subset of all elements x $\in$ T with the property P(x),

Type and domain constuctors

bool	boolean variables
sca	scalar
ma	matrix
boolmat	boolean matrix
unseq	unique sequence (sequence without repeating)
seq	sequence
set	intset- a set of integers realset- a set of real values
graph(U,W)	graph with the vertex set U and arc set W
fun T → Q	function from T in(to) Q
bifun	biection
surfun	surjection

infun	injection
$\cup$	set union
$\cap$	set intersection
\	set subtraction
$\triangleq$	"by definition"- operator

Selected keywords:

"corresponding to", "associated with", "let", "is".

Elementary examples.

Example 1.

There are 6819 TV sets in a warehouse. 516 must be shipped to one store and 293 - to another. How many TV sets will be left in the warehouse after these shipments?

W, S1, S2, S3 $\triangleq$ set

TVS $\triangleq$ sca

!W.*TVS=6819

!S1.*TVS=516

S2.*TVS=293

?S3.*TVS where (S3 $\cup$ S1 $\cup$ S2=W) & (S1 $\cap$ S2=∅).

Example 2.

To maximize :

$$5x_1 + 6\bar{x}_2 x_3 + 2\bar{x}_1\bar{x}_4 + 8\bar{x}_3\bar{x}_4 \to \max$$
$$2x_1\bar{x}_2 + 6\bar{x}_3 + 4x_3\bar{x}_4 < 6$$
$$4\bar{x}_1 + 3\bar{x}_2 + 5\bar{x}_3 + \bar{x}_4 > 4$$

X $\triangleq$ set, Z $\triangleq$ set

X=$\{x_1, x_2, x_3, x_4\ \}$

Z={0,1}

F$\triangleq$ surfun X → Z

?F where
$(\max(5x_1 + 6\bar{x}_2 x_3 + 2\bar{x}_1 x_4 + 8\bar{x}_3\bar{x}_4))\&((2x_1\bar{x}_2 + 6\bar{x}_3 + 4x_3\bar{x}_4) < 6)\ \&$
$((4\bar{x}_1 + 3\bar{x}_2 + 5\bar{x}_3 + \bar{x}_4)>4\ ).$

Example 3.

Let the following graph be given

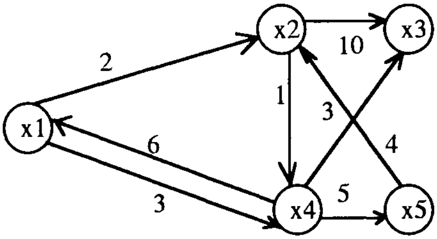


Fig. 5.4.

It is required to find the minimum total length path connecting vertices 1 and 3,

T,U,W $\triangleq$ set i,j $\triangleq$ sca

PATH $\triangleq$ seq

GPAPH(T,U)
i,j ∈ {0,1,...,N}
T={x1,x2,x3,x4,x5}
U={u(1,2),u(2,3),u(2,4),u(1,4),u(4,1),u(4,5),u(5,2),u(4,3)}
W={2,10,1,3,6,5,4,3}

F $\triangleq$ bifun U → W

where U corresponding to W

$\mathrm{PATH}(\alpha,\beta) \triangleq \mathrm{unseq}\ \{u(i,j)\ \mathrm{where}$

$(u(\alpha,\_) \in \mathrm{PATH})\&$

$(u(\_,\beta) \in \mathrm{PATH})\&$

$(*u(i,j).!\ u(j,k)\ \mathrm{where}\ j \neq \beta)\}$

$?\mathrm{PATH}(1,3)\ \mathrm{where}$

$(\min(\Sigma(r\ \mathrm{where}\ (r=F(t))\&(t \in \mathrm{PATH}))).$

The following are equivalencies on the set of primitive relations:

(i) $\neg A \sim \#(A)$

(ii) $A \vee B \sim (\#A)\#(\#B)$

(iii) $A \multimap B \sim \mathrm{right}(A)\#B$

(iv) $A \,\&\, B \sim \#(A\#B)$

(v) $a{:}b \sim a\#b$

(vi) $A \rightarrow B \sim A\#\overline{B}$

(vii) $A \succ B \sim \mathrm{right}\ (B)\#A$.

Consider now language L2. It includes the following conceptual operators.

(i) **find**(x|P(x)), where P(x) is a property of an object x formalized by means of the predicate language and functional analysis.

Examples:

$\mathrm{find}(x|x^2-2x+1=0)$

$\mathrm{find}(x|x+\sin x=1/2)$

$\mathrm{find}(x,y|x+y=2,x-5y=8)$

$\mathrm{find}(x1,x2,x3,x4|x1\#x2,\ \#(x3,x2,x4),\ \#(\overline{x}2,x3))$.

Obviously, this operator may be adequately expressed in L1, e.g.,

$x \triangleq \mathrm{sca}$

$?x\ \mathrm{where}\ (x+\sin x=1/2).$

(ii.a) **create_problem** (situation, conditions, criterion)

(ii,b) **create_subproblem** (problem, situation, conditions, criterion)

Example.

>create_problem (situation S0;

f0,A1,B1,C2,B2,f1,f2 $\triangleq$ bool

f0=A1~C2
f1=B1~¬B2
f2=¬A1v¬B1vB2&C2;
criterion
f0 & f1 & f2=TRUE)

(iii) **test_validity** (x $\triangleq$conditions; situation; criterion)

Example. Assuming the previously created situation S0 we have

>test_validity(x$\triangleq$A1 & B2; situation S0;

criterion x=TRUE)
>VALID

(iv) **derive_consequences**(situation,conditions,goal)

Example

>derive_consequences (situation S0; ; goal *)
/* this command produces logical consequences from S0 */
>A1,$\overline{\text{B1}}$,B2,C2
$\overline{\text{A1}}$,$\overline{\text{B1}}$,B2,$\overline{\text{C2}}$
etc.

(vi) **change_situation** (situation 1: conditions 1;
situation 2: conditions 2)
situation 1 is replaced by the situation 2

(conditions 2 are introduced instead of conditions 1)

(vii) **create_computation_patterns** (x1,x2,...,xn|y1,y2,...,yn)

Let x1,x2,...,xn be media objects and for every object x_i the corresponding procedure(-res) which calculate x_i is known. Thus, designation

$$x_i = \varphi_i(x_{i1}, x_{i2}, \ldots, x_{it}) \qquad (*)$$

is used do denote that x_i is calculated by procedure φ_i with inputs $x_{i1}, x_{i2}, \ldots, x_{it}$. Let, for instance

$$\begin{aligned}
&\varphi_1 \colon < x_1, x_3 | x_4 > \\
&\varphi_2 \colon <| x_1, x_2, x_5 > \\
&\varphi_3 \colon < x_2, x_5 | x_7, x_9 > \\
&\varphi_4 \colon < x_2, x_7 | x_4 > \qquad (**) \\
&\varphi_5 \colon < x_1, x_3, x_4 | x_6, x_8 > \\
&\varphi_6 \colon < x_3, x_6 | x_5, x_7 > \\
&\varphi_7 \colon < x_5, x_6 | x_9 >.
\end{aligned}$$

Here a vertical bar is used to separate inputs and outputs of the procedure. The system (**) is called a computation pattern.This is used for automatic equation synthesis on the basis of the procedure given in chapter 3. To represent a computational pattern a kind of graphical editor is used which enables the creation of a hierarhical tree for the problem. Let there be the following problem: "to find the sum S,

$$S = \left[\frac{n+1}{2}\right] + \left[\frac{n+2}{2^2}\right] + \left[\frac{n+2^2}{2^3}\right] + \ldots + \left[\frac{n+2^k}{2^{k+1}}\right] + \ldots$$

for the given integer n>0. Here [x] denotes the greatest integer value t such that t≤x, e.g. [3.21]=3, [4.09]=4 , etc. A computational pattern for this problem is shown in Fig.5.5.

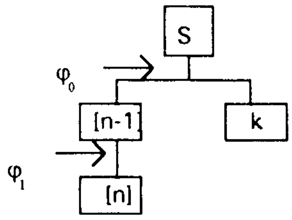


Fig. 5.5

φ_0 and φ_1 are represented as follows:

$$\varphi_0 \overset{\Delta}{=} S = \left[\frac{n}{2}+\frac{1}{2}\right]+\left[\frac{n}{2^2}+\frac{1}{2}\right]+\ldots+\left[\frac{n}{2^{k+1}}+\frac{1}{2}\right]+\ldots$$

$$\varphi_1 \overset{\Delta}{=} \left[r+\frac{1}{2}\right]=[2r]-[r]$$

n,r are free variables. The resulting value of S is computed on the basis of the substitution ,φ_1 for φ_0, which gives

$$S=[r]-\left[\frac{r}{2}\right]+\left[\frac{r}{2}\right]-\left[\frac{r}{2^2}\right]+\ldots0+0-0+\ldots$$

or S=[r]=r, where r=n.

(viii) **use_substitution** (situation, context1, context2).

This operator provides a unification of the two contexts.

Example 1.

>use_substitution (*; x^2 - x + 2;

x $\overset{\Delta}{=}$ $y^{1/2}$)

>y-$y^{1/2}$ + 2

Example 2.

>use_substitution(*;

P(X,f(X),a);

$\overline{\mathrm{P}}$(a,u,W))

X=a, u=f(a), W=a

(ix) **define_rule** (semantic specification).

This conceptual operator takes task's concept specification (see section 0.5.2), for instance:

```
>define_rule(
< MA (binary, symmetrical) >
< RD2 >
< RSP1, RSP7, RSP9 >
< SS1 >
< ER2 >
< RR3 >
< SP5 > ).
```

This enables us both to define the type of a problem within the classification available and determine a method of solution or a heuristic. Thus, the answer may be as in the following

> This is a TA-problem: maximum zero-submatrix of a given 0,1- matrix. There is a number of solving methods: TA.M1, TA.M2, TA.M15, TA.M24. Forc further information see the corresponding method depiction.

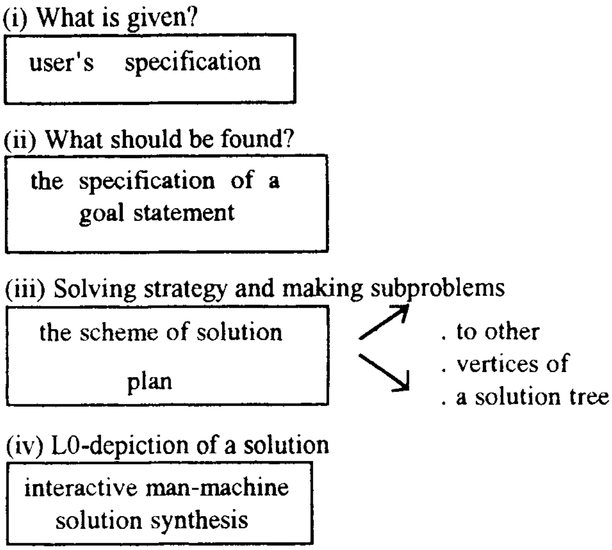
(x) **create insertion point** (name)

(xi) **delete insertion point** (name)

(xii) **run** (name, inputs, outputs).

These operators will be discussed later.

Finally, consider language L0. As it was pointed out earlier this is a form of the user dependent language without a strictly outlined syntax. L0 uses graphical editors to represent a solution synthesis as a growing tree. Each vertex of the tree begins with the standard frames:



(i) What is given?

user's specification

(ii) What should be found?

the specification of a goal statement

(iii) Solving strategy and making subproblems

the scheme of solution plan

. to other
. vertices of
. a solution tree

(iv) L0-depiction of a solution

interactive man-machine solution synthesis

Thus, as we can now see, languages L0,L1,L2 correctly correspond to the programming paradigm of **CAPSS** we deal with here. Namely, L0 and L2 provide manipulations with a problem concept by a human-solver and L1 is used to specify the formal task to the computer-solver. L0 is a user-defined language providing context-free manipulations of a problem. On the contrary, L2 is a context-dependent language with a formally defined syntax: It enables the user to create such mechanisms as an intelligent oracle and insertion points,to perform context unification and to make equations.

5.7.2. An intelligent oracle

An intelligent oracle represents a form of the logical prover which gives one of the three possible answers to the user's questions: "VALID (TRUE) INVALID (FALSE) and INDEFINITE (FAIL TO ANSWER)". Consider the following logical puzzle "In the wood there live knights, liars and sorcerers. Knights tell only truth, liars tell lies and a sorcerer may be either a knight or a liar. One meets persons A, B, C.

It is known that one of them is a sorcerer. A says that C is a sorcerer. B says that he himself is not a sorcerer. And C says : "At the least two of us are liars". The question is who is the sorcerer (a knight or a liar)?

First, the user creates the problem:

```
>create_problem (situation S0;
f0 = A1 ~ C2
f1 = B1 ~ ¬ B2
f2 = C1 → ¬A1 & ¬B1 v ¬C1 & ¬A1 v ¬C1 & ¬B1 ;
f3 = A2 & ¬B2 & ¬C2 v ¬A2 & B2 & ¬C2 v ¬A2 & ¬B2 & C2
/* x = A,B,C
x1 = 1 if x is a knight and
x1 = 0 if x is a liar ;
x2 = 1 if x is a sorcerer, and
x2 = 0 otherwise */
criterion: f0 & f1 & f2=TRUE).
```

Then the user may use the following fragment of interactive dialog with the oracle:

```
>test_validity (x ≜ A2 & A1 ; situation S0;
criterion x=TRUE).
>INVALID /* A2 & A1 is FALSE */
>test_validity (x ≜ ¬A2 & A1 ; situation S0 ;
criterion x=TRUE)
>INVALID
>test_validity (x ≜ A2 & ¬A1 ; situation S0;
criterion x=TRUE)
>VALID
>test_validity(x ≜ B2 & ¬B1 ; situation S0;
criterion x=TRUE)
>VALID
```

Thus, the user is able to test the validity of his conjectures. However, he is not thoroughly confident in the consistency of the initial model S0. He may establish inconsistency (derivability of the formula Z & ¬Z) by means of the operator "derive_consequences (situation, conditions, goal)", for example,

```
>derive_consequences(S0;*;*)
```

This command will produce all the interpretations valid in S0:

```
>A1,B1,¬C1,¬A2,¬B2,C2
¬A1,¬B1,¬C1,¬A2,B2,¬C2
¬A1,B1,¬C1,A2,¬B2,¬C2 (etc.).
```

The user may change the problem specification, for example

```
>change_situation (situation S0:
f0 = A1 ~ C2
f1 = B1 ~ ¬B2;
situation S0:
f2 = C1 → ¬A1 & ¬B1).
```

This operator results in adding a new formula f2 to the "old" formulas f0 and f1. It is not difficult to observe that, for the redefined situation s0, the user needs to make a greater amount of assumptions to get the answer "Sorcerer is a liar". This example demonstrates that there is a real share of responsibilities between the human-solver and a system. Evidently, that oracle is not restricted to logical puzzles. The concept outlined here is a programming pattern which integrates a wide field of logical and analytical problems.

5.7.3. Insertion points

To clarify the notion of the insertion points consider an example. Suppose,the human-solver attempts to find a sum

$$S = \left[\frac{n+1}{2}\right] + \left[\frac{n+2}{2^2}\right] + \ldots + \left[\frac{n+2^k}{2^{k+1}}\right] + \ldots$$

(all the designations are the same as in section 5.7.1).

The following is a possible L0-specification of a solution depiction

$$S=\left[\frac{n+1}{2}\right]+\left[\frac{n+2}{2^2}\right]+\ldots+\left[\frac{n+2^k}{2^{k+1}}\right]+\ldots+0\ldots$$

$$\left[x+\tfrac{1}{2}\right]\neq[x]+1$$

$$[x+y]\geq[x]+[y]$$

$$[x-y]\leq[x]-[y]$$

>create_insertion_point (A1)

I $\underline{\Delta}$ intset

R $\underline{\Delta}$ realset

S,x $\in$ R

l,n,k $\in$ I

F $\underline{\Delta}$ infun R $\rightarrow$ I

where F(x) = max(n where n $\leq$ x)

Q $\underline{\Delta}$ bifun I $\rightarrow$ I

where $Q(n)=\sum\left(F\left(\frac{n+2^k}{2^{k+1}}\right)\right)$

where

k = min(l where $F\left(\frac{n+2^l}{2^{l+1}}\right)=0$)).

The user may now use A1 as a subprogram:

```
run (A1,5,x).

x=5.

run (A1,68,x).

x=68.

run (A1,321,x).

x=321.
```

From this example the user can guess that the sum $S(n)=n$. To prove the fact ,he may create another insertion point

```
>create_insertion_point(A2)
define_rule(_).
run (A2,
<
<SE (functional sequence)>,
<RD1>,
<RSP5>,
<RR7>,
<ER1>,
<SS4>,
<SP1>>,_ ).
```

(i). One may use the scheme of inductive proof in the form

T(n) is supposed to be TRUE

(ii). One may try to represent the functional sum in the form

$FSum=f(0)-f(1)+f(1)-f(2)+f(2)...$ e.t.c.

(iii). One may see some analogies with the typical examples.

(iv). One may try to apply the Z-transform method. For more information see the corresponding method depiction.

Suppose, the user has decided to attempt case (ii). The system prompt may be as follows.

Theorem. If function $\varphi(x)$ satisfies the condition

$$\varphi(x+1)-\varphi(x)=f(x)$$

then it is correct that

$$f(0)+f(1)+...+f(n-1)=\varphi(n)-\varphi(0).$$

Using this latter formula the user will possibly try to establish the relationship between

$[x+\frac{1}{2}]$ and $[x]$.

This relationship is a key factor in the resulting solution.

Thus, we can see that insertion points enable the user to establish an interface with formal methods and system-oriented procedures. These may be user–defined subprograms and functions, rule specifications, and an intelligent helper (not considered here).

5.7.4. Exhaustive search procedure

The operator "**find** (x|P(x))" may be directly realized on the finite domain D ,x ∈D with the small cardinality |D|. This may be possible due to the resolvability of the formal constraints P(x). Thus, for the problem

$x \underline{\Delta}$ set, $Z \underline{\Delta}$ set

$x=\{x_1,x_2,x_3,x_4,x_5\}$

$Z=\{0,1\}$

$F \underline{\Delta}$ surfun $x \rightarrow Z$

?F where

$((x_1\bar{x}_2 \vee x_3)\&(\bar{x}_1 \vee \bar{x}_3 \vee x_2x_4)\&(\bar{x}_4 \vee \bar{x}_1x_3)$=TRUE))

the system may produce the following sequential solution procedure:

(i) $x_1=0$ (randomly generated value)

$x_3 \cdot 1 \cdot (\bar{x}_4 \vee x_3)$=TRUE (reduced constraint)

(ii) $x_3=0$ (randomly generated value)

0=TRUE (fail)

$x_3=1$ (alternative assignment)

1=TRUE

$F \underline{\Delta} <x_1=0, x_3=1>$

Yet another example:

U={u(1,2), u(2,3), u(2,4), u(1,4), u(4,1), u(4,5), u(5,2), u(4,3)}

CYCLE $\underline{\triangle}$ unseq {u(i,j) where

(*u(i,j).! u(k,j))&

(u(k,i) ∈ PATH)&

(u(i,a) # u(i,b) where α≠β)}

?CYCLE

The solution process resembles the previous one in the main features :

(i) u(1,2) (first choice)

(ii) {u(1,2), u(4,1)} (second choice)

(iii) {u(1,2), u(4,1), u(2,4)}

>CYCLE={u(1,2), u(4,1), u(2,4)}.

5.8. Making a semantic structure of the problem

Let there be given an L1-specification of the problem with the form

X,Z $\underline{\triangle}$ set

X={x_1,x_2,x_3,x_4}

Z={0,1}

F:surfun X → Z

?F where

(max($5x_1 + 6\bar{x}_2 x_3 + 2\bar{x}_1 x_4 + 8\bar{x}_3 \bar{x}_4$)).

Now the question becomes how to solve this? Is it necessary to have a kind of universal solving procedure (for example, in terms of Tarsky's theorem [68])? However, we adhere to a specialized problem-dependant approach. The next step we are to realize is that of making a semantic structure of the above L1-specification. Fortunately, there are a number of available conceptual frames for building such a structure. The reader is now referred directly to section 0.5 (for problem conceptualization). The frame language outlined there is quite suitable for our purposes. The result of this conceptualization is shown below:

We assume the availability of an appropriate interpreter realizing mapping L_1-specification → Frame_based_semantic_representation. To continue these considerations we assume also that **CAPSS** is specialized on the definite form of a mathematical problem (in our case it is the discrete optimization area). A very advantageous feature of NP-complete problems is their reducability to each other. This opens us good possibilities for realizing a solving strategy based on the following scheme.Each method (procedure, heuristic, rule) is specified in the frame knowledge base by means of a semantic pattern SP_i. We directly associate SP_i with a general task concept structure (Fig. 0.6) where the task is considered to be an input L_1-specification unifiable with SP_i. The method i is considered suitable for the problem j if

SP_i (is a generalization of) L^1_j

where L^1_j - is a L_1 - specification of the problem j. We shall use a simpler notation for this relation , namely

x ♣y

will denote that y is a realization of x.

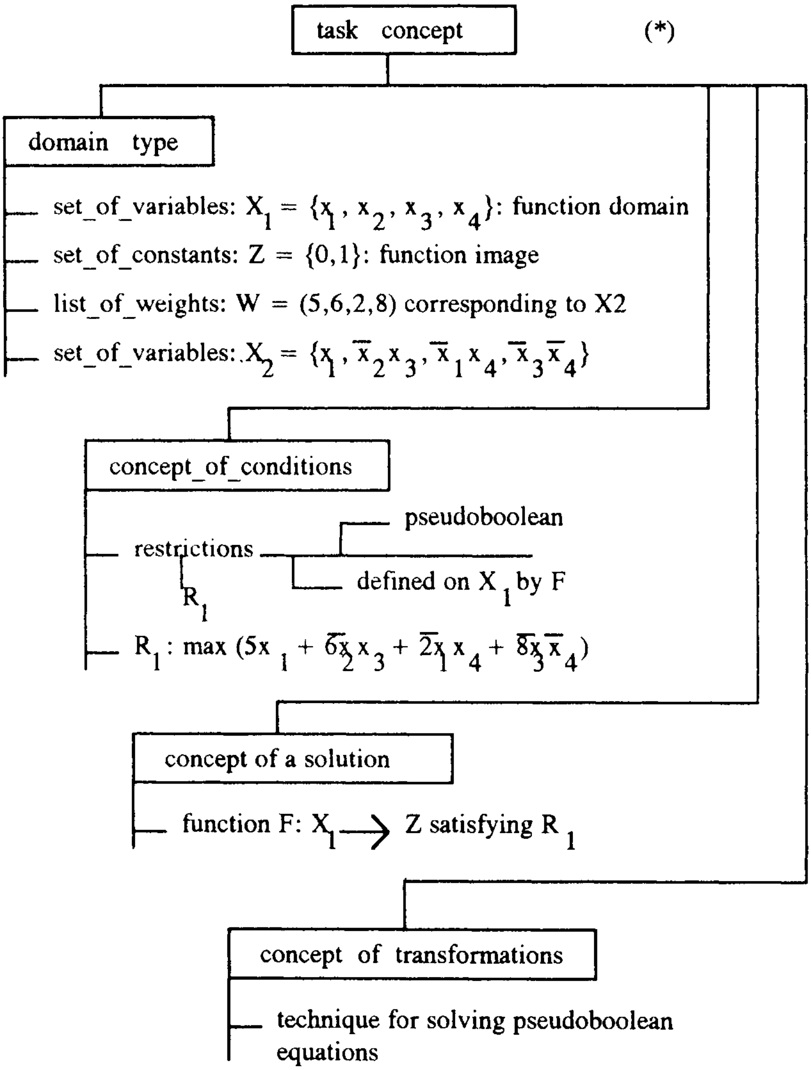
Two main problems are associated with ♣ :

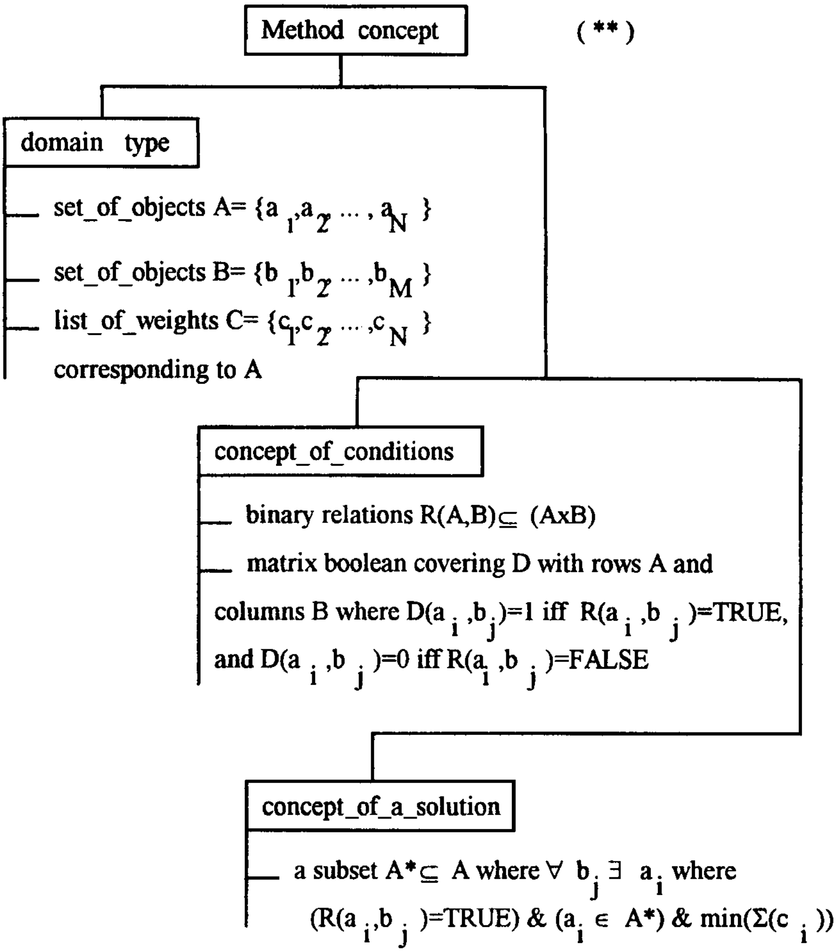
(i) pattern unification, and

(ii) pattern transformation.

The problem of the pattern unification is the same as the problem of the term unification in Prolog. The main difficulty is connected with point (ii), i.e. pattern transformation. It is, therefore, necessary to give some explanation of this..

To apply this method to the problem with the previously defined task concept it is necessary to use a domain transformation. Consider the following two concepts as an illustration.


task concept
(*)
domain type
set_of_variables: $X_1 = \{x_1, x_2, x_3, x_4\}$: function domain
set_of_constants: $Z = \{0,1\}$: function image
list_of_weights: $W = (5,6,2,8)$ corresponding to X2
set_of_variables: $X_2 = \{x_1, \bar{x}_2 x_3, \bar{x}_1 x_4, \bar{x}_3 \bar{x}_4\}$
concept_of_conditions
restrictions
pseudoboolean
defined on X_1 by F
R_1
R_1 : max $(5x_1 + \overline{6x}_2 x_3 + \overline{2x}_1 x_4 + \overline{8x}_3 \bar{x}_4)$
concept of a solution
function F: $X_1 \longrightarrow Z$ satisfying R_1
concept of transformations
technique for solving pseudoboolean equations



Method concept (**)

domain type

- set_of_objects A= $\{a_1, a_2, \dots, a_N\}$
- set_of_objects B= $\{b_1, b_2, \dots, b_M\}$
- list_of_weights C= $\{c_1, c_2, \dots, c_N\}$ corresponding to A

concept_of_conditions

- binary relations $R(A,B) \subseteq (A \times B)$
- matrix boolean covering D with rows A and columns B where $D(a_i, b_j)=1$ iff $R(a_i, b_j)=\text{TRUE}$, and $D(a_i, b_j)=0$ iff $R(a_i, b_j)=\text{FALSE}$

concept_of_a_solution

- a subset $A^* \subseteq A$ where $\forall\ b_j\ \exists\ a_i$ where $(R(a_i, b_j)=\text{TRUE})\ \&\ (a_i \in A^*)\ \&\ \min(\Sigma(c_i))$

The following transformations are a part of the allowable homomorphisms defined on the different task domains:

graph (oriented, weighted) ↔
matrix (boolean, asymmetrical) & list (of weights)
equation (pseudo boolean) ↔
matrix (boolean, symmetrical) & list (of weights)
equation (logical) ↔
matrix (boolean, symmetrical)
system (of logical equations) → equation (logical)

matrix (boolean, symmetrical) ↔ matrix (boolean, covering)

system (of algebraic equations) →

equation (polynomial with one unknown)

restriction (pseudoboolean) ↔ equation (pseudoboolean)

restriction (algebraic) → equation (algebraic with

additional unknown)

equation (algebraic, linear) → hyperplane.

Is it possible to unify the concepts given by (*) and (**) with the help of the homomorphisms listed above? As we know, this is not a simple unification but a solution scheme with an indirect application of the method (*) to the problem (**).

Consider the homomorphism

equation (pseudoboolean) ↔

matrix (boolean, symmetrical) & list_of_weights.

To be useful, this scheme should be realized as below:

equation (pseudoboolean) $\{:(w_1v_1^* + \ldots + w_nv_n^*) \rightarrow \frac{\max}{\min}$

set_of_variables $V1=\{v_1,\ldots,v_n\}$

set_of_constants $Z=\{0,1\}$

list_of_weights $W=\{w_1,\ldots,w_n\}$ corresponding to V2

set_of_variables $V2=\{v_1^*,\ldots,v_n^*\}\} \rightarrow$

matrix (boolean, symmetrical) {: MAT with rows V2 and columns V2

where

MAT $(v_i^*, v_j^*)=0$ iff $v_i^* \& v_j^* \neq$ FALSE

MAT $(v_i^*, v_j^*)=1$ iff $v_i^* \& v_j^* =$ FALSE } &

list_of_weights {: $W = \{w_1,\ldots,w_N\}$ corresponding to V2 }

The following homomorphism should be applied next.

matrix (boolean, symmetrical) $\leftrightarrow$

matrix (boolean, covering)

which has the following concrete representation:

matrix (boolean, symmetrical) {: MAT with rows V2 and

columns V2 where

MAT (v_i^*,v_j^*)=1 iff v_i^*& v_j^* $\neq$ FALSE &

MAT (v_i^*,v_j^*)=0 iff v_i^* & v_j^* = FALSE } $\rightarrow$

matrix (boolean, covering) {: MAT1 with rows V2 and

columns V3 where

!j. MAT1 (l,j) = MAT1 (i,j) iff MAT (v_i^*,v_l^*)=1

MAT1 (i,j)=0 iff ($\exists$ l,m MAT1(l,j) = MAT1 (m,j)=1)&

(i$\neq$l)&(i$\neq$j).

Thus, the last homomorphism enables us to get the following representations of the concepts in (**), i.e.,

concept_of_conditions

— binary relation R(V2,V3)Δ MAT1(i,j)=1 where i$\in$ V2, j $\in$ V3

— matrix (boolean, covering) MAT1 with rows V2 and
columns V3 where

!j. MAT1(l,j)=MAT1(i,j)=1 iff (MAT(v_i^*,v_l^*)=1) & (i<l))

MAT1(i,j) iff ($\exists$ l,m MAT1(l,j) = MAT1(m,j)=1) & (i$\neq$l) & (i$\neq$j)).

domain type

— set_of_objects V2= $\{v_1^*, \dots, v_n^*\}$

— set_of_objects V3= $\{j_1, \dots, j_m\}$

— list_of_weights W= $\{w_1, \dots, w_n\}$ corresponding to V2

concept_of_a_solution

— a subset $V_2^* \subseteq$ V2 where

$\forall\ j \in V3\ \exists\ i \in V_2^*$ where

$(\mathrm{MAT1}(i,j)=1\ \&\ (\min\Sigma(c_i)))$

Thus, after unification with task concept (*) we have

$X_1=\{x_1,x_2,x_3,x_4\}$ unified with $V1=\{v_1,v_2,v_3,v_4\}$, N=4

$Z^{(*)}=\{0,1\}$ unified with $Z^{(**)}=\{0,1\}$

$W^{(*)}=\{5,6,2,8\}$ unified with $W^{(**)}=\{w_1,w_2,w_3,w_4\}$

$/w_1=5,\ w_2=6,\ w_3=2,\ w_4=8/$

$X2=\{x_1, \bar{x}_2 x_3, \bar{x}_1 x_4, \bar{x}_3 \bar{x}_4\}$ unified with

$V2=\{v_1^*, v_2^*, v_3^*, v_4^*\}$

$/v_1^* \triangleq x_1; v_2^* \triangleq \bar{x}_2 x_3; v_3^* \triangleq \bar{x}_1 x_4; v_4^* \triangleq \bar{x}_3 \bar{x}_4\ /$

$$\mathrm{MAT}=\begin{array}{c|c|c|c|c|} & v_1^* & v_2^* & v_3^* & v_4^* \\ \hline v_1^* & 0 & 0 & 1 & 0 \\ \hline v_2^* & 0 & 0 & 0 & 1 \\ \hline v_3^* & 1 & 0 & 0 & 1 \\ \hline v_4^* & 0 & 1 & 1 & 0 \\ \hline \end{array} \qquad \mathrm{MAT1}=\begin{array}{c|c|c|c|} & j1 & j2 & j3 \\ \hline v_1^* & 1 & 0 & 0 \\ \hline v_2^* & 0 & 1 & 0 \\ \hline v_3^* & 1 & 0 & 1 \\ \hline v_4^* & 0 & 1 & 1 \\ \hline \end{array}$$

From this example (with many of the unnecessary details omitted) the reader may draw the conclusion that an effective resolution strategy is required for the ♣-relation. Its main features are similar to that used for the compilation of computer programs [19].

5.9. Conclusion

Three categories of languages have been introduced and considered in this chapter : L0, L1 and L2. Their role may be represented schematically by Fig. 5.6.

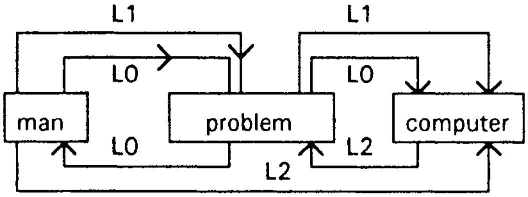


Fig. 5.6.

The computer is regarded as an ally with the human-user and the special notion of an insertion points has been intentionally introduced to denote the interface L0 →L1 and L0 →L2. The language L1 has been defined which enables the declarative specifications of a problem to be made as an exact formal system. L1-specification is translated into a semantic structure by means of the language L2.

It has been shown that each mathematical method is provided with an appropriate semantic depiction SP_i. So, to use method i for the problem j, a resolution strategy is required to establish that

$SP_i \clubsuit (L2\text{-}j)$,

i.e., that SP_i is a generalization of the L2-specification corresponding to the problem j.

Yet another important mechanism referred to as an intelligent oracle has been described. This mechanism essentially employs a theorem proving technique to produce three types of the answers "VALID", "INVALID", and "INDEFINITE".The user is entirely responsible for the correct model representation and its completeness.

The whole approach to the **CAPSS** organization described above takes into consideration all the essential ideas of the sharing of responsibilities between the human-solver and the computer. To realize this approach we have introduced languages L0, L1 and L2 (Fig.5.6.) and defined the following specialized tools:

- intelligent oracle
- insertion points
- weak method specification as a subprogram (function) in programming languages
- a universal weak operator

find(x|P(x)).

This Page Intentionally Left Blank

Chapter 6

FUTURE CONCEPTS: SOME PHILOSOPHICAL ISSUES

This is the final chapter of the book where an attempt will be made to outline possible future directions in problem solving. An approach exists [69] which explains future achievements by means of the state of theoretical investigations 20 - 30 years before the analysed period. Thus, to predict the state of the problem solving field in 1995 - 2000 it is necessary to return to the 1965 - 1975's investigations. Evidently, these terms (20 - 30 years) are rather subjective but the whole idea seems to be sound as it stems from the position of historical determinism.

6.1. Universal problem solving approach restoration

Attempts are now being undertaken to restore a universal solving strategy, especially in the area of weak methods and heuristic reasoning. Naturally, they cannot be regarded as accidental phenomena. The main idea of a universal approach is for the generalization of distinct concrete algorithms which are relevant to the problems from a given domain. This notion of universality is much narrower than Leibniz' original presumption. There exist algorithmically insoluble problems that are the main obstacles in universal approach realization. The following theorems serve as illustrations of such a situation.

Theorem 6.1. *[70]. There is no algorithm which defines for any other algorithm whether it stops or not for the given input data.*

Theorem 6.2. *[70]. The problem of deducibility in first order logic is insoluble.*

Theorem 6.3. *[70]. The problem of word equivalence in associative calculus is insoluble.*

The famous ancient problem of the trisection of an angle is of a similar kind. A particular case of this problem for $\varphi = 90°$ is, however, resolvable by means of compasses and ruler. The solution is illustrated by Fig.6.1, where BN=BM=NQ=MR=r and r is the radius of the circle with the centre in the point B. The problem which is insoluble in the given formal system may be resolvable in another formal system containing the former one. Thus, the trisection problem may be resolved by means of auxiliary tools using special geometrical ideas. One of such an idea is the notion of quadratrix.

Consider Fig. 6.2. Suppose radius r rotates around point A with the velocity $w = \frac{\varphi}{t}$

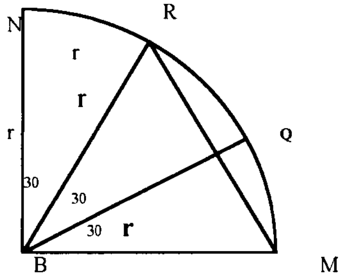


Fig. 6.1.

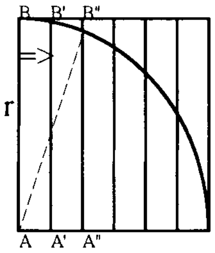


Fig.6.2

Simultaneously, the line AB is shifting to the right with the velocity $v = \frac{l}{t}$. Then the quadratrix is the line which connects the intersection points of the radius r and the line AB in its progression (A'B'; A".B" ,etc.). (Fig. 6.3).

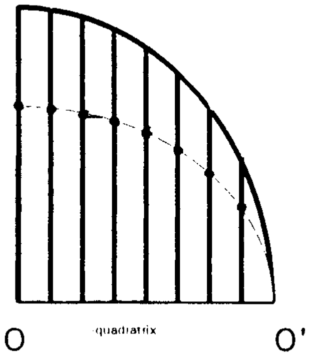


Fig. 6.3.

Now it is clear, that if the quadratrix is given, the problem of trisection is reduced to dividing a given side (OO' in Fig. 6.3.) into 3 equal parts. This example shows the possible issues of an insoluble problem:

(i) there may exist some particular variants which are effectively resolvable;

(ii) to obtain a solution one needs auxiliary means, not represented in the theory, which are used to formalize the problem.

It is, however, not clear whether such additional means are always possible. Evidently, deductive reasoning is insufficient to constitute a universal solving approach as it does not provide an extension of problem formulation. However, our hopes of finding a solution in non-deductive reasoning are fraught with the same defect. This fact immediately follows from the next theorem.

Theorem 6.4. *[71]. There exists a totally defined incalculable function.*

This theorem has far-reaching consequences in logic and problem solving. Thus, we cannot seriously consider a universal strategy paradigm without imposing strict constraints on its applicability. Therefore, we should define more precisely the conditions providing universal approach feasibility. Our expectations are connected, first of all, with the application of the universal approach to find elements of a solution. Evidently, they may or may not be relevant to the final result. This means that "universal mechanisms" are needed primarily not as comprehensive solvers but as auxiliary tools for extending a problem's frames. It is important in this context to define what form of universal mechanisms are more suitable from computational and cognitive viewpoints.

We are witness to different conceptual approaches for developing this paradigm. Perceptrons, heuristic evaluation functions and heuristic programming, generalized patterns of weak methods, GPS-paradigm by Newell, Simon and Shaw, intelligent automation and psychological models of cognition are good explanatory examples.

The common feature of these paradigms is in their presentation of some basic universal scheme modeling cognitive processes. No matter how this scheme is realized, it may be regarded as an intelligent virtual process (VP) specializing in definite problem solving activities. The structure of VP includes the following components:

- interpreter (I)
- analyzer (A)
- body of VP (B)
- planner (P).

The interpreter provides communication between the different VPs. It recognizes relevant information and parses it in order to get an internal problem specification. Thus, regarding language L1 outlined in the previous chapter, one may draw a conclusion that the interpreter should create a corresponding L2-specification suitable for further processing.

The analyzer is responsible for forming a correct estimate of a VP's capabilities to solve a problem. Evidently, it should be able to define the type of problem and choose correctly a method or an available algorithm. It is assumed that the analyzer and the planner communicate strongly with each other because the latter is engaged in process planning and generating subgoals. The analyzer and the planner may sometimes be considered as a single module or even a definite part of the body of the VP.

The body of a VP as its main part is a set of general solving activities incorporating basic weak methods. It should be noted here that we do not require the VP's body to be extremely universal, for example, as the models of MacCulloch & Pitts. An approach by Lauriere [5.4] in Alice and by Seidel in [5.5] is more preferable due to taking into account problem's domains peculiarities.

As it was noted earlier, one of the basic VP's functions is **"find (x|P(x))"**. We suppose that it should get more serious attention to the developing of the universal paradigm. Evidently, to be of practical importance P(x) should be efficiently resolvable. The above mentioned works of Lauriere and Seidel may serve as proof of that statement. Thus, summing up the main issues of a universal-strategy concept ,the following deserves attention.

(i) the universal approach in problem solving is justified mainly for its ability to extend formal representation by discovering hidden regularities in the problem's domain. It scarcely may be regarded a serious competitor to effective formal methods. However, there is a class of problems with no formal searching algorithms or with only inefficient solving procedures. So, to raise the efficiency of the universal approach it is reasonable to take into account peculiarities of the domains and use powerful heuristics, for example, as those in the constraint satisfaction method of Laurie.

(ii) it follows from the above that a generalized pattern of solution techniques should incorporate a set of efficiently resolvable properties P(x) to enable usage of **"find (x|P(x))"** - based weak methods.

(iii) the whole universal paradigm may be represented by the framework of an intelligent virtual process (VP) and its corresponding communication scheme. It comes down to a representation of the following general form

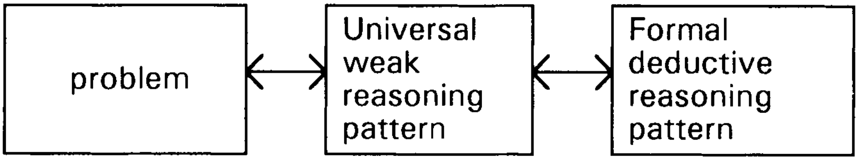

problem
Universal weak reasoning pattern
Formal deductive reasoning pattern

Fig. 6.4.

Each known paradigm of the universal solving approach lies within the boundaries of the VP-concept. However, it should be noted that the more primitive a VP organization and a mathematical representation are, the more complicated should be the mutual VP's interconnection and specialization. This implies, by the way, that it is impossible on the basis of primitive mathematical formalizm alone to create a powerful universal solver. It seems that the problem of the co-operation of a very large number of primitive solvers is, as we suppose, the main difficulty in the realization of universal strategy approach.

6.2. Weak methods become strong

Evidently, we are interested in making weak methods strong enough to produce efficient solutions of the problems. This requires us to develop suitable searching patterns, for instance, involving cut technique. One needs to consider the triad <**P**, **Path**, **Solution**> where **P** is a problem, **Path** represents a solution process for the problem, and **Solution** is the resulting interpretation representing the answer. The main peculiarity of the considered pattern is that **Path** need not be an optimal one. It further means that weak methods suffice to get the **Path** provided that there is a sufficient cutting algorithm. Therefore, one may cut either **Path** or **Solution** to repeat the solution process. However, this scheme requires any criterion for the solution's optimality. As the reader may note, this pattern was applied to the NP-complete problems considered in the chapter 1. We suppose that its efficiency is rather high enabling its wide utilization in application systems. This paradigm suitability was also demonstrated through the Φ-transform method in discrete optimization problems.

Evidently, there are other patterns such as branches-and-boundary method(s) and A^* - algorithm utilizing weak methods. So, we may draw a conclusion that further developments in the weak methods area are connected with extending formal application schemes by formalizing different searching patterns.

The other important issue is connected with utilizing expert systems. This paradigm incorporates weak mechanisms into the solving policy based on the knowledge available. The expert system (ES) is valuable both as a problem solving system paradigm and a programming concept directly utilizing weak methods. In our opinion, ES may be regarded as a new generation of problem solving system noteworthy for its orientation to the problems with *a priori* unknown solving algorithms. Present investigations in the theory of ES, however, have concentrated mainly on the formal issues of knowledge representation and knowledge processing. Meantime, the tuple "human-problem" as a component of triad "human-problem-computer" is not paid such an attention as it deserves. Evidently, when solving a problem human needs in different languages to reach different goals. The following languages are required:

- for the formal representation of problems;
- for manipulating the concepts of a task;
- for specifying solving activities (heuristics, intelligent oracle, weak methods, rules for cuts, etc.) and using them as procedure calls;
- for intelligent help;
- for creating internal concepts of a problem.

The realizing of the family of languages provides extension of the primary ES framework as it assumes the involvement of the human-solver in the solution process. Furthermore, it requires, in our opinion, a new approach in the learning to solve problems as a kind of speciality. Obviously, there are two types of metaprinciples - universal and specific ones. As an example of the latter metaprinciple one may consider the principle "divide and conquer". Cantor's diagonal method is a good example of a specific theoretical rule which is widely used in proving algorithmic insolubility and set non-recursiveness. The conclusion may be drawn that to use a weak method effectively the human-solver should know how to interpret it in the given problem's conditions. Thus, the learning in problem solving is directed to the recognition of problem patterns and suitable solution schemes. This problem is far from being entirely resolved.

To sum up the above considerations the following should be noted:

- there exist formal patterns providing the utilization of weak methods to obtain a required solution. One such pattern was considered in this book, namely: "use a weak method to obtain a good approximation to the required solution. Cut either the solution obtained or the way it was found. Repeat the scheme until the criterion of optimality becomes satisfied". There is a number of practically efficient algorithms with that paradigm, viz. the well-known Dantzig's algorithm in linear programming, branch-and-bound method by Little et.al., Gomory's algorithm in integer programming ,etc.

- in a human-machine problem solving system the human-solver should be able to interpret weak principles within a given problem's frame. Evidently, his skill is the subject to appropriate training and learning. In **ES** paradigm we can see a direct utilization of knowledge which is available for the solver though a human-solver is not immediately engaged in the solution process. As a theoretical issue of the matter it is required to provide formal ways of weak principles interpretation. However, this problem is still far from being solved.

6.3. The role of formal logic in future developments

Logic has a great impact on the theory and practice of problem solving. Thus, our expectations are strongly connected to further developments in this area. Consider some important issues relevant in this context.

It is a well-known fact that logic is a semi-resolvable system due to Gödel's incompleteness theorem. The question may be raised of the conditions disabling this theorem. In the paper [5] a general attempt has been made to introduce formal systems (called S-systems) possessing the desired property.

Let S be a formal system in the language L(S) incorporating arithmetic.

Let $s([\varphi].x)$ be a term in L(S) such that $s([\varphi],\bar{n})$ defines the **Gödel** number of the proposition $\varphi(x)$ provided that x is replaced by $\bar{n}$. Finally, let Pr_s (.) be a predicate stating that the **Gödel** number of the formula $\varphi(x)$ is $s([\varphi],x)$. S-system is defined as the system where one or more conditions from those given below is violated:

G1) for each proposition σ in L(S) if $S\vdash\sigma$ then $S\vdash Pr_s([\sigma])$;

G2) for each proposition σ from L(S) if $S\vdash Pr_s([\sigma])$ then $S\vdash Pr_s([Pr_s[\sigma]])$;

G3) for every two propositions σ_1 ,σ_2 :

$S\vdash PR_s([\sigma_1])$ & $PR_s([\sigma_1\rightarrow\sigma_2]) \Rightarrow PR_s([\sigma_2])$.

Now, the following questions are valid:

- to point out at least one S-system with a non-empty class of S-problems S-problem is defined as a formula φ(x) with one free variable x which satisfies the following conditions:

$$\mathrm{S}\vdash\left\{\begin{array}{c}\forall x(\mathrm{Pr}\ (s[\varphi],x))\vee \mathrm{Pr}_s(s[\overline{\varphi}],\varphi)));\\ \forall x(\mathrm{Pr}_s(s[\varphi],x))\rightarrow \varphi(x));\\ \forall x(\mathrm{Pr}_s(s[\overline{\varphi}],x))\rightarrow \overline{\varphi}(x)).\end{array}\right\}$$

- for an arbitrary S-system to define a class of S-problems.

Let us further proceed to the following issues of formal logic which are relevant to problem solving.

(i). A theorem proving technique may be regarded as a kind of universal weak method. All the above mentioned ideas concerning the strengthening of weak principles has a direct link with inference strategies in logic. This especially concerns optimization problems formalized in logic or by means of logic. Many well-known NP-problems and the other problems in graph theory provide a good approbation field for logical methods. An interesting area of applied logical systems is the scheduling theory. As an example consider the following problem formulation.

Let for the jobs X = {x_1, x_2, x_3, x_4, x_5, x_6} the following relations are fulfilled:

(x_1#x_3) (x_4#x_3)

(x_1#x_6) (x_5#x_3)

(x_2#x_4) (x_5#x_6)

(x_2#x_5)

where (x_i#x_j) means that the jobs x_i and x_j cannot be performed by the same machine. Suppose, there are two machines. Let t_1=2, t_2=5, t_3=6, t_4=2,t =2, t_6=4 be the processing times. It is necessary to divide jobs into two subsets X_1 and X_2 to provide

$$\max\left(\sum_{x_j\in X_1} t_j;\ \sum_{x_k\in X_2} t_k\right)\rightarrow \min$$

There are no incompatible pairs ,either in X_1 or in X_2. This scheduling problem may be reduced to the pseudoboolean optimization problem as follows.

Find an interpretation providing

$$(2x_1\bar{x}_3 + 6\bar{x}_1x_3 + 2x_1\bar{x}_6 + 4\bar{x}_1x_6 + 5x_2\bar{x}_4 + 2\bar{x}_2x_4 + 5x_2\bar{x}_5 + 2\bar{x}_2x_5 +$$
$$6x_3\bar{x}_4 + 2\bar{x}_3x_4 + 6x_3\bar{x}_5 + 2\bar{x}_3x_5 + 2x_5\bar{x}_6 + 4\bar{x}_5x_6 - -\frac{1}{2}\sum_{i=1}^{6} t_i)^2 \rightarrow \min, \qquad (*)$$

i.e., the total processing time of every set (X_1 and X_2) should be reduced from $\frac{1}{2}\sum_{i=1}^{6} t_i$ to as small as possible. Evidently, the solution of (*) defines either set X_1 or X_2 only. It means, that this solution does not warrant that the other set will be compatible. To meet this requirement let us introduce the constraint:

$$(x_1 \vee x_3)(x_1 \vee x_6)(x_2 \vee x_4)(x_2 \vee x_5)(x_4 \vee x_3)(x_3 \vee x_5)(x_5 \vee x_6) \equiv 1 \qquad (**)$$

which warrants the suitability of both sets provided that (**) is satisfied.

The problem above is resolved by the solution

$$X_1 = \{x_2, x_3, x_6\}, X_2 = \{x_1, x_4, x_5\}.$$

(ii). Logic has "penetrated" into the fields of mathematics which were traditionally "non logical", e.g. linear integer programming [75], combinatorial optimization [76] and others. It is realized by means of suitable logic-oriented languages which are used for both problem representation and solution derivation. The latter possibility is, however, restricted by Gedel's incompleteness theorem. Let us review a general scheme of solution making by a proving technique:

(i) formulation of computability theorem in the form ∀x∃y f(x,y) (for every valid inputs "x" there exist corresponding outputs "y" computed by program f(.))

(ii) proving computability theorem

(iii) extracting program (algorithm) from the proof.

Evidently, Gedel's theorem concerns p.(ii) of the above scheme. However, there are definite difficulties connected with p.(iii) as well. It is known from pure mathematics that so-called existence theorems have no constructive proofs. In this case p.(iii) of the solution scheme is not realizable. As an example, consider the following problem: to prove that there are two irrational numbers a and b such that a^b is a rational number Let $\sqrt{2}^{\sqrt{2}}$ be irrational, then assuming a=$\sqrt{2}^{\sqrt{2}}$ and b=$\sqrt{2}$ are the desired numbers as $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = 2$. On the contrary, if $\sqrt{2}^{\sqrt{2}}$ is rational then a=$\sqrt{2}$ and b=$\sqrt{2}$ are desired numbers as well. Evidently, this proof does not state irrationality or rationality of $\sqrt{2}^{\sqrt{2}}$. therefore, it cannot be considered a constructive one. Using logic in theoretical and applied problems is becoming more profound. Two main aspects are the subject for further investigations:

- developing different schemes of logical reasoning and applying logical methods in various mathematical fields;
- applying of logical formalizm in problem stating in the input languages of computer- aided problem solving systems.

6.4. The human factor

It was many times stated in this book that human is considered a decisive factor in problem solving process. His role is extremely high in the problem formulation phase, dividing the problem into subproblems and making purposeful subgoals. Considering this role of the human-solver we, evidently, make computer-aided problem solving system (**CAPSS**) man-dependent. It means that the efficiency of **CAPSS** will vary correspondingly to the abilities and the qualification of users. It may be found reasonable to build **CAPSS**'es with different orientation and qualification levels. From the very abstract viewpoint there are two basic classes of problems: the ones using "try-and-test"-principle and the others mainly dependent on "insight" and "gestalt"-concepts [78]. In fact, every difficult creative task requires both of these approaches. We adhere to an approach which distinguishes between different levels of **CAPSS** specialization. *0-level* is formed by basic **CAPSS** with a general orientation, for example, to making equations [78] and solving logical puzzles [79]. *1-level* is formed by the systems using theorem proving technique and weak methods with indistinct specialization. These systems are more powerful and capable of solving complicated problems in an appropriate area providing automatic solution synthesis.

The following levels are represented by the specialized problem solving systems (for example, ALICE - in combinatorial optimization or MACSYMA in computer mathematics, etc.) The specialization is necessary whenever the system operates with difficult and serious problems.

When considering human factor one should delimit the area of human interests according to his participation in the solution process. Thus, we obtain the following main fields where the human factor is essential.

- Problem raising.
- Problem formulation.
- Searching for algorithm.
- Reviewing and correcting factors relevant to points 1-3.

Problem arising. It is essential what kind of problems are put forward and why. On the one hand, this issue is important in connection with **CAPSS** efficiency. On the other hand, the laws of the problems arising are important as they are. It suffices to point out eminent Hilbert problems formulated on the 1'st Mathematical Congress (Paris, 1900). In this connection, the problem raising may be considered as the reflection of a working mathematician on his work. It is a deep psychological phenomenon which needs wide investigation. Evidently, the theoretical backgrounds of purposeful behaviour are directly connected with this issue.

Problem formulation. Very often problems cannot be solved because of inappropriate formulation. To eliminate possible misunderstanding, a system should supply the user with the necessary means to facilitate the formalization process (for example, usage of animation as in [78] is suitable for this purpose). A graphical editor enabling the creation of a problem's structure seems to be a useful tool as it realizes the conception of structured programming in problem solving. Thus, the user is engaged in the process of creating the problem structure as a whole which facilitates his perception of a problem and decreases the possible difficulties in finding a solving procedure. Consider an illustration: "Two cars have started traveling towards each other from points A and B.

The distance S_A =100 km. It is known that the first car has a speed which is higher than that of the second car by 20^{km}_{h} . The cars meet 2 hours after starting out. What is the distance covered by each car if they have been moving uniformly."

The problem structure is shown in Fig. 6.5.

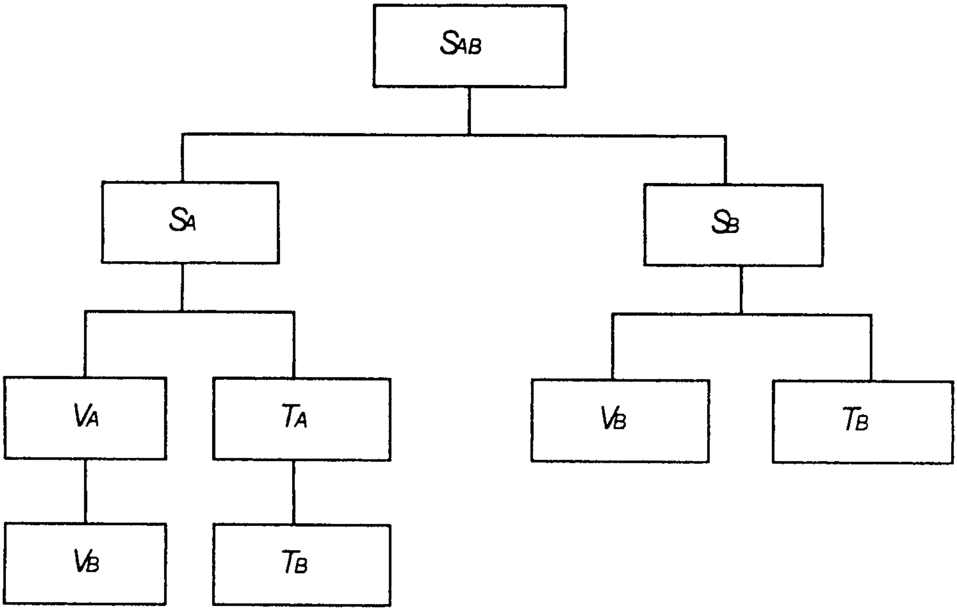


Fig. 6.5

$V_A(V_B)$ denotes the speed of the car which started from A(B) and $T_A(T_B)$ is the time elapsed until meeting.

Every fragment of the structure (as in Fig. 6.5) defines some equation, i.e.

1. $S_{AB}=S_A+S_B$
2. $S_A=V_A*T_A$

3. $S_B=V_B*T_B$
4. $T_A=T_B$
5. $V_A=V_B+20$

If this system is resolvable then it may be reduced to the root-equation dealing with one variable only. Thus, in our case

$S_A=(V_B+20)*T_B$

$S_B=V_B*T_B$

$S_{AB}=2V_B+2V_B+40$

$V_B=15$

$V_A=35$

Note, that the root-equation making is performed in the bottom-up direction of the problem structure.

Searching for an algorithm and correcting the searching process

The main issue of the topic is how to organize the searching activity to provide its high efficiency. There is a fine analogy with inventive activity in engineering, where such methods were devised by H.Altshuller, as brain storming or algorithms for solving inventive tasks. There are also interesting works by H. Poincare and G. Birkhoff concerning this issue. The main problem identified by these authors is connected with the difficulties of investigating subconsciousness and related issues. Evidently, we are still far from having a mechanical representation of the intellectual activity of brain. Psychology gives us important facts about indirect influence of external and internal factors on brain productivity. The following factors are recognized to be of great importance:

- deep "loading" into the problem;
- high interest in problem solution;
- attention and ability to concentrate efforts on one subject;
- patience;

- competition with other people;
- dissatisfaction with current situation.

These factors have proved to be of great importance for the quality of human solving activities. However, it is necessary to clarify the role of computer in increasing the brain productivity. How may the computer stimulate the human-solver to find good solutions? It seems, however, that there are no direct answers to this question. The main activities are concentrated around the organizational help provided by the intelligent computer.

6.5. Are there other paradigms ?

The reader could note that our considerations were connected mainly with two basic paradigms in problem solving: the "universal" weak strategies and theorem proving techniques. Indeed, these two paradigms make a great impact on problem solving theory despite some serious restrictions which have been stated earlier. Now it is reasonable to ask if any other(s) suitable paradigm(s) exist(s). The answer, as we understand it, may be positive if the analogy with human thinking is taken into consideration. Indeed, such mechanisms as imagination and intuition are very important for a creative mind. Unfortunately, there is a lack of suitable formal theory to describe models of intuition and relevant issues. However, we place our hopes on further success in this field.

We pointed out earlier that an effective paradigm will not be elementary whatever similarity with brain functional organization is used. It seems impossible (or at least too difficult) to realize a complicated and powerful artificial problem solving system based on primitive basic elements. In this connection our expectations are based on the advanced schemes such as, for instance, considering a virtual solving process. This also implies that we should consider the models of co-operating virtual processes and, therefore, interaction concepts are of great importance to us. It is necessary to recognize that interaction of intelligent processes is a relatively new theoretical branch with interesting perspectives for the entire problem.

One of the promising directions in studying problem solving mechanisms consists of the differentiation of the relevant common operations in the solving processes. In [80] the following common operations are put forward: proceduralization, composition, generalization, and discrimination.

Proceduralization is the common operation to produce relevant "*How*-knowledge" from the "*What*-knowledge". Thus, the following production:

G1 **IF** the goal is to create a structure and there is an operation that creates such a structure

THEN use that operation

may be transformed to the procedure in a suitable context.

If, for example, the goal is to infer formula f from the given set of formulae $F(F \vdash f)$ then G1 transforms to a more specific goal description:

G2 **IF** the goal is to infer f from F

THEN use inference rules (such as linear resolution strategy).

Composition enables us to get new productions as a combination of the others.

Let "**IF** C1 **THEN** A1" and "**IF** C2 **THEN** A2" be a pair of productions to be composed, where C1, C2 are the conditions and A1, A2 are the corresponding actions. Then their composition is "**IF** C1 & (C2 - A1) **THEN** (A1 - G(C2) & A2". C2 - C1 denotes the conditions of the second production not satisfied by structures created in the action of the first.

Generalization and discrimination. The notions of generalization and discrimination may be simply introduced by means of the following examples [80]. Consider the following productions:

IF the goal is to generate the present tense of Think

THEN Write Think + S

IF the goal is to generate the present tense of Thank

THEN Write Thank + S

Generalizing these rules, we obtain

IF the goal is to generate the present tense of X

THEN Write X + S

where X is a free variable.

On the contrary the discrimination enables one to distinguish between mutually incompatible cases of a given phenomenon, e.g.

IF the goal is to generate the present tense of X and the subject of the sentence is singular

THEN Write X + S

IF the goal is to generate the present tense of X and the subject of the sentence is plural

THEN Write X

These discrimination and generalization mechanisms are very much like knowledge acquisition mechanisms that have been proposed in the artificial intelligence literature. Thus, summing up our considerations, the following issues deserve to be mentioned:

- human's abilities such as imagination and intuition should be modeled effectively and an appropriate formalized theory is required to develop problem solving paradigms.
- elementary basic concepts of solving schemes are supposed to be insufficient and more complicated models are needed.
- theoretical interests should be shifted to the problems of the interaction between elementary solvers in their co-operation and specialization.

6.6. Conclusive remarks

In this book an attempt has been made to develop theoretical and programming issues of problem solving. The former aspect of the problem was considered in a somewhat different way from the traditional viewpoints.

We mean that our notion of weak methods includes not only rule-based searching strategies but also a number of mathematical schemes with a high degree of abstraction.

It especially concerns the approach considered in this book, where it is of primary importance in understanding properly the relations between solving operations. Indeed, in traditional approaches the main point of interest is connected with applying rules to the given states. They merely concern the relationship between rules and states as the subjects of investigation. On the contrary, in mathematics it is essential to take into account interrelations between solving operations and the structure of the problem as a whole. Thus, our theoretical interests were mainly connected with the structure of the mathematical task and possible general schemes of solution applicable to this structure.

One of the inferences from this investigation may be made as follows: there are effective and efficient schemes enabling weak methods to become strong. It is obvious that developing this concept is of great interest to mathematicians. Two such schemes were shown in the book dealing with discrete optimization problems.

As far as the programming paradigm in problem solving is concerned we have suggested several languages enabling us to formulate problems, modify their structures, and to build efficient mechanisms such as an intelligent oracle and an intelligent helper. It is obvious now, that since **CAPSS** incorporates human and machine activities, the languages are required to provide interaction in the following directions:

(i) human ↔ problem

(ii) computer ↔ problem

(iii) human ↔ computer

Obviously, the form of language for the issues (i, ii, iii) is drastically different from traditional forms. We would like to note that to be efficient , the language should be oriented to the definite class of mathematical problems. In our case such an area was defined in discrete optimization and logical problems. Thus, it became possible to apply specialized weak methods with their semantics compatible with problem concepts.

We consider human to be the main factor in a solution searching process. Therefore, he (she) needs languages and an appropriate support in the problem formulation phase and the algorithm developing phase also. The role of the computer is essential in providing formal solution synthesis and tedious calculations.

REFERENCES

1. Tyugy E.-H. Conceptyalnoe programmirovanie. - Moscow, Science, 1984 (in Russian).

2. Tyugy E.-H., Harf M.Ya. Algorithmy structurnogo synthesa program. // Programmirovanie, N 4, 1980, p.p. 3-13 (in Russian).

3. Demetrovics J., Knuth E., Rado P.. Computer - Aided Specification Techniques. // Computer and Automation Inst., Hungarian Academy of Sciences, 1989, P.114.

4. Liskov B., Guttag J.. Abstraction and Specification in Program Development. - The Mit Press, Mc.Graw-Hill Book Company, New York, 1985.

5. Nilsson N.J. Principles of Artificial Intelligence.- Tioga publishing company, Palo Alto, California, 1980.

6. Korf R. Learning to solve problems by searching for macro-operators. - e.a. Pitman Adv. Publ. Program, 1985, p.p. 1-147.

7. Dowsing R.D. A first course in Formal Logic and its applications in computer science - Blackwell Scientific Publications, 1986, P. 266.

8. Kowalski R. Logic for problem solving. - Elsevier, North Holland Inc., 1979.

9. Chang CH-L., Lee R.C-T. Symbolic logic and mechanical theorem proving. - Academic press, New York - San Francisco - London, 1973.

10. Beth E.W. The foundations of mathematics, Amsterdam, 1959.

11. Robinson G.A., Wos L. Paramodulation and theorem proving in first order theories with equality.// In Machine Intelligence, 4/Ed. B.Meltzer and D.Michie, N.Y.: American Elsevier, 1969, P. 135-150.

12. Sibert E.E. A machine oriented logic incorporating the equality relation.// In Machine Intelligence, 4/Ed. B.Meltzer and D.Michie. N.Y.: American Elsevier, 1969, P. 103-134.

13. Lloyd J.W. Foundations of Logic Programming. - Springer - Verlag, Berlin-Heidelberg-New York-Tokio, 1984.

14. Apt K.R. Introduction to Logic Programming - Report CS-R8741, Centre for Mathematics and Computer Science, 1009 AB Amsterdam, the Netherlands, p.p. 1-55, 1986.

15. Slagle James. Studies in computer science.//Studies in Mathematics, 1986, Vol.22, p.p. 229-279, Washington University.

16. Banerji Ranan. et al. Theory of problem solving.//Proc. IEEE", 1982, vol. 70, N 12, p.p. 1428 - 1448.

17. Paradimitriou Ch., Steiglitz K. Combinatorial optimization: algorithms and complexity. - Princehton University, Inc. Englewood Cliffs, 1982.

18. Lawler E.L. Recent Results in the theory of Machine Scheduling.// Mathematical Programming: The State of the Art. Edit. by A.Bachem, p.p. 202-234, Bonn, 1982.

19. Aho A., Ullman J.. The theory of parsing, translation and compiling. vol. 1,2. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1972.

20. Hilbert D., Bernays P. Grundlagen der mathematik. 1 - Springer-Verlag, Berlin, 1968.

21. Thayse A., Gribomont P. et al. Approche logique de l'intelligence artificielle. 1 De la logicue classique a la programmation logique. - Dunod informatique, Paris, 1988.

22. Kaufmann A., Henry-Labordere A. Methodes et modeles de la recherche operationnelle. Paris-Bruxelles-Montreal, 1973, vol. 2.

23. Lawriere J-L. Intelligence artificielle. - Resolution de problemes par l'Homme et la machine. - C.F.PICARD-PARIS, 1986.

24. Swamy M.N.S., Thulasiraman K. Graphs, Networks and Algorithms. John Wiley & Sons, New York, 1980.

25. Reingold E.M., Nievergeld J., Deo N. Combinatorial algorithms: Theory and Practice. - Prentice-Hall, Inc., Englewood Cliffs, 1977.

26. Deo N. Graph Theory with Applications to Enginneering and Computer Science, Prentice-Hall, Englewood Cliffs, N.J., 1974. 27. Harary F. Graph Theory. Addison-Wesley, Reading, Mass., 1969.

28. Hadamard J. Essai sur la psychologie de l'invention dans le domaine mathematique. - Paris, Librairie scientifique, 1959.

29. Polya G. Mathematics and plausible reasoning. vol. 1. - Princenton univer. press. Princenton, 1954.

30. Chicinadze V.K. Reshenie zadach nelineynoi optimizacii: metod Ψ-preobrazovania.- Nayka, Moscov, 1983 (in Russian).

31. Lasserre J.R. An analytical expression and algorithm for the volume of a convex polyhedron in R. // J. Optim. theory and appl., 1983, vol.39, N 3, p.p. 363-377.

32. Algorithmy i programmy dla vosstanovlenia functionalnych zavisimostey./ Edit. V.N. Vapnick, Moscow, Science, 1984, 516 p.p. (in Russian).

33. Boender G., Rinnooy Kan A.H.G. e.a. Astochastic method for global optimization. // Mathem. programming.- 1982. vol.22, N 2, P. 125-140.

34. Vapnick V.N. Vostanovlenie zavisimostey po empiricheskim dannym. Moscow, Science, 1979, 448 p. (in Russian).

35. Luice K.D. Prognostic methods for economic issues (Russian edition). Moscow, Finance & Statistics, 1986, 133 p.

36. Gardner E.S., Dannenbring D.G. Forecasting with exponential smoothing: some guidelines for model selection. // Decision Sci., 1980, vol.11, N 2, p.p. 370-383.

37. Jain C.L. Smoothing constant key to exponential smoothing.// J. Bus. Forecast., 1983, vol.2, N 3, p.p. 24-25.

38. Seljutin V.A. Machinoe proektirovanie elektronnych ustroystv. Moscow, Soviet Radio, 1977.

39. Sysoev V.V., Perov V.A. Razrabotka programm mnogokriterialnoy optimizacii na base Ψ-preobrazovania. // Economics & Applied mathematical methods. 1976, vol.12, N 1 (in Russian).

40. Sysoev V.V. Avtomaticheskoe proektirovanie promyslennogo konveyera i industrialnogo oborudovania v microelectronnom proizvodstve. Moscow, Radio & Svyaz, 1982 (in Russian).

41. Sukharev A.G. et al. Metody optimizacii. Moscow, Science, 1983 (in Russian).

42. Pshenichny B.N. Metody linearizacii. Moscow, Science, 1983 (in Rissian).

43. Hardy G.G. Inequalities. Moscow, Foreign Literature, 1948, 456p.

44. Gallab A., Allard M. In "Modern languages & logic programming". Proc. 1th Int. Workshop in Rennes, North-Holland, 1984.

45. Vhang Ling, Vhang Bo. The statistical inference method in heuristic search . In "Modern languages & logic programming". Proc. 1th Int. Workshop in Rennes, North-Holland, 1984, pp. 757-759.

46. Krakowiak S. Principes des systemes d'exploitation des ordinateurs. - Dunod Informatique, 1986.

47. Davies D.W. et al. Computer networks and their protocols. - John Willey & Sons, New York - Toronto, 1980.

48. Aho A., Hopcroft J.E., and Ullman J.D. The design and analysis of computer algorithms. Addison-Wesley, Reading, Mass., 1974 (VAM).

49. Computer and job-shop scheduling theory. /Ed. by E.G.Coffman, Jr. - John Willey & Sons, 1976.

50. Uvarov S.I. Issledovanie odnoy problemy raspisania na grafe. // Automatics & Telemechanics. Moscow, N 7, 1985. p.p. 172- 175 (in Russian).

51. Garey M.R., Johnson D.S. Computers and intractability: A guide to the theory of NP-completness. San-Francisco. W.H.Freemen & Company, 1979.

52. Barsky A.B. Vvedenie v teoriy parallelnych vichisleniy. Moscow, Mashinostroenie, 1980 (in Russian).

53. Bellman R., Kalaba R. Dynamic programming and modern control theory. Academic Press, New York & London, 1967.

54. Kotva M. Testing of simulation model validity: methodological problems. // Syst. Anal. Model. Simul. 5(1988), 4, p.p. 393-402.

55. German O.V. Multiprocessornye raspisania v odnorodnych vychislitelnych sistemach. // Automatic and computing technics. (Riga, Latvia), N 5, 1985, p.p. 70-77 (in Russian).

56. Phillips D.T., Garcia-Diaz A. Fundamentals of network analysis. Prentice-Hall Inc., Englewood Cliffs, 1981.

57. Newell A., Shaw J.C., Simon H.A. Empirical explorations of the logic theory machine: a case study. Report P-951, Rond. Corp., 1957, March.

58. Wang Hao. Toward mechanical mathematics. // IBM J. Res. Devel, 4, N 1, 1960, p.2-22.

59. Glushkow V.M. Kapitonova Yu., et al. Postroenie yazykov matematicheskich teoriy. // Cybernetics, N 5, 1972, p.p. 19-28 (in Russian).

60. Ko Sakai. Toward mechanization of Mathematics - Proof Checkers and Term Rewriting System. // Programming of Future Generation Computers. / Ed. by K. Fuchi, N. Nivat. Elsevier Sci. Publishers, 1988, p.p. 335-390.

61. Laird J.E., Newell A. A universal weak Method. Carnegie- Mellon Univ., 1983.

62. Rayward-Smith V.J., McKeown G.P., Burton F.W. The general problem solving algorithm and its implementation. // New Generation Computing, 6, 1988, p.p. 41-66. Springer-Verlag and OHMSHA LTD.

63. Lauriwre J.-L. Intelligence artificiell. Rwsolution de problemes par l'Homme et la machine. Troisieme edition, Eyrolles, Paris, 1988.

64. Davenport J., Siret Y., Tournier E. Calcul formel. Sustwmes et algorithmes de manipulation algebriques. Masson, Paris-N.Y. 1987. 65. Bhla J. The evolution methods in problem solving. // Advances in Modelling & Simulation, AMSE Press, vol. 16, N 3, 1989, p.p. 49-64.

66. Bongard M. Problema rasposnavania. Moscow, Science, 1967 (in Russian).

67. Havranwk T. Formal systems for data analysis. // International Journal of Man-Machine Studies, 1981.

68. Tarski A. A decision method of elementary algebra and geometry. - Berkeley: Univ. of California Press, 1951.

69. Expert systems. Principles and case studies. / Ed. by R.Forsyth. Chapman & Hall, London, 1984.

70. Kleene S. Introduction to methamathematics. D. Van Nostrand Company Inc., USA, 1952.

71. Cutland N. Computability. An introduction to recursive function theory. Cambridge Univ. Press, 1980.

72. Lauriere J.L. A language and program for stating and solving combinatorial problems. // Artif. Intel., vol. 10, 1978, p.p. 29 - 127.

73. Seidel R. A new method for solving constraint satisfaction problems. // Proc. IJCAI (Vancouver, Canada, 1981), p.p. 338-342.

74. Ershow Yu.L., Samokhvalow K.F. Novy podchod k filosofii matematiki. // Computer Systems, vol. 101, 1984, p.p. 141-148 (in Russian).

75. Mc. Kinnon K.I.M., Williams H.P. Constructing integer programming models by the predicate calculus. // Annals of Operat. Research, 21 (1989), p.p. 227-246.

76. P. Van Hentenryck. A logic language for combinatorial optimization. // Annals of Oper. Research, 21 (1989), p.p. 247- 274.

77. Sackman H. Man-computer problem solving. // Auerbach Publishers Inc., N.Y., 1970.

78. Nathan M.J. et al. An unintelligent tutoring system for solving word algebra problems. Proc. IFIP TC. 5 Conf. CAD/CAM Technol., Transfer - Mexico, 1988.

79. Valentine M., and Davis R.H. The automated solution of logic puzzles. // Information Processing letters. 24(1987), p.p. 317 - 324.

80. Anderson J.R. A theory of the origins of Ruman knowledge. // Artificial Intelligence, vol.40, N 1-3, 1989, p.p. 313-351.

This Page Intentionally Left Blank

GLOSSARY

1. **Algorithm** - mathematical formalism regarded as an exact and deterministic procedure which can be realized in the form of a computer program. An exact notion of Algorithm is connected with the notion of Turing machine. (Rather often Algorithm is associated with a mapping A→B where A, B are discrete sets of sentences over corresponding languages).

2. **Algorithmically solvable problem** - a problem which can regularly be solved by means of some algorithm ,i.e. it may be formulated and written as a sentence X of input language A of some Turing machine which transforms X into the sentence Y of the output language B where Y represents a solution of the problem { see items 1, 3, 40, 51 of Glossary for the related information }.

3. **Algorithmically un(re)solvable problem** - a problem which cannot be (re)solved by means of some algorithm, i.e. there is no suitable algorithm at all {1, 51 }.

4. **"Black Box"** - an entity with unknown inner nature which can be studied by exploring the links between its inputs and outputs.

5. **Boolean variable** - a variable which can take only two values, e.g. "0" and "1" {19}.

6. **CAPSS** - computer- aided problem solving system. A system specializing on solving problems by means of the human-machine interaction. There is a number of CAPSS paradigms establishing different conceptual approaches to the roles of a human and a computer in the solving processes.

7. **Church Thesis** - an assertion which states that an intuitive and formal notions of computability are equivalent.

8. **Clause** - a formula of the form

B(b1,...,bn) :-

$A(a1,...,am)$,

. . . .,

$C(c1,...,ck)$.

corresponding to a disjunct {19, 39, 49} :

$\overline{A}(a1,...,am)\vee...\vee\overline{C}(c1,...,ck)\vee B(b1,...,bn)$

9. Complete problem - a problem X such that any other problem Y can be reduced to X provided that X and Y belong to the same class of problems.

10. Combinatorial exploison - a situation which is characterized by an exponential growth of the number of possible interpretations for the problem X under consideration when its length is increased linearly {24}.

11. Completeness - a property of a given formal theory TH with a language L consisting in the fact that for each formula $\varphi \in L(TH)$ either φ or not-φ ($\overline{\varphi}$) can be proved in TH.

12. Complexity function of an algorithm - a function which measures time (corresponding memory sizes) required by an algorithm with respect to the length (n) of the task specification given in input (formal) language of some Turing machine {36}.

13. Concept - a semantic structure which comprises the notions linked in one of the following ways:

- as a function (predicate) and its arguments;
- as an action and its subjects (objects);
- as a causa and its consequences.

14. Consistency - a property of a given formal theory TH implying that there is no any formula $\varphi \in L(TH)$ such that both φ and not-φ are provable in TH {11, 15, 18, 25, 36}.

15. Contradiction - a situation when one demonstrates that some formula and its negation both can be proved in a given theory {14, 18, 25, 36}.

16. Criterion - a restriction on the solution in the form of a function, or an equality, or an inequality {51}.

17. Cutting strategy - a solving strategy excluding the parts of a search tree which do not contain an optimal (or exact) solution {45, 51, 54}.

18. Deducibility - a relation between the formulas **f1,...,fn** regarded as premises and a formula h such that there exists a logical inference of the formula h from

f1,...,fn

realized by means of inference rules. To prove a formula means to *deduce* it from the given set of formulas.

19. Disjunct - a boolean formula of the form {5,17}

$$(x_{i1}^{\alpha 1} \vee x_{i2}^{\alpha 2} \vee \ldots \vee x_{iz}^{\alpha z})$$
$$x_{ij}^{\alpha j} = x_{ij} \quad if \quad \alpha j = 1$$
$$x_{ij}^{\alpha j} = \overline{x}_{ij} \quad if \quad \alpha j = 0$$

20. Domain - a set of all possible values for the problem variables {51}.

21. Efficiency (of an algorithm) - means that an algorithm has a polynomial complexity {12}.

22. Equivalency of the problems A and B - means that A can be (polynomially) reduced to B and vice versa {52}.

23. Existence theorem - a theorem of the form

$$\forall x \exists y \varphi(x,y)$$

asserting that for any inputs x there exist outputs y calculated by means of some procedure φ(x,y).

24. Exponential algorithm - an algorithm with the complexity function of the form $O(2^n)$, n!, $2^{nLog\ n}$, etc (that is, non-polynomial complexity function) {12, 48}.

25. Formal system - a mathematical structure of the form

< L, R, IR >

where: **L** is the language of a system;

R are the rules for making well-formed formulas (correct formulas);

IR are the inference rules.

26. Formal theory - is the same as a formal system {25}.

27. Functor - a term of the form

f(t1,...,tn),

where **f** is a function symbol and **t1,...,tn** are the terms {59}.

28. Gödel's incompleteness theorem - the theorem asserting that there exist incomplete theories containing arithmetic of the wholes {34}.

29. Heuristic algorithm - an algorithm which uses some principle or rule(-s) to find a solution of the problem which, however, may not be optimal {1, 40, 47, 51}.

30. Heuristic evaluation function - a function h(x) estimating a lower bound of the number of nodes in the search tree to be open from the node x before the final node will be reached 54}.

31. Hypothesis P≠NP - states that each NP-complete problem (e.g. Minimum-size cover problem) cannot be resolved on the basis of polynomial algorithm {12, 24, 43, 44, 48}.

32. Implication - a formula of the form X→Y equivalent to not-X ∨ Y.

33. Inference rule - a scheme of the form

$$\frac{f_1, f_2, \dots, f_k}{\varphi}$$

interpreted as follows:" if each formula $f_1,\dots,f_k$ is proved then φ should be considered proved formula {18, 25, 26}.

34. Incomplete theory - a theory TH in which some formula cannot be proved or refuted {25, 28, 33, 36}.

35. Interpretation - a set of variable bindings suiting a given model {51}.

36. Language - a mathematical structure of the form

L(TH)= < A, O, R>$_{\mathbf{TH}}$

where:

A is an alphabet, i.e. a set of symbols and auxiliary signs (such as brackets,commas, etc.)

O are the primary objects of the theory **TH** such as terms and formulas;

R is a set of the rules for producing so-called well-formed expressions (sentences) of the language L;

TH is a theory which consists of the sentences of the language **L** which are called therems {25,26}.

37. Logical approach to problem solving - an approach based on the theorem proving technique in which a problem is regarded as a theorem of some formal theory and a solving procedure deals with proving this theorem {23, 26, 33, 39}.

38. Logical inference - a kind of reasoning which uses the logical inference rules {33, 50}.

39. Logic programming - programming based on theorem proving technique {8, 18, 33}.

40. Meta-procedure - a procedure which is based on weak methods. Meta-procedure comprises a number of principles showing how to use weak method(-s) to obtain an exact (optimal) solution of the problem in a practically admittable way {6, 17, 61, 62}.

41. Model - a number of relations between a problem's objects {20,51}.

42. Monotonicity - a property of the heuristic evaluation function h(x) based on the following relationship:

$h(x) \geq h(y)$ if and only if node x preceds node y in a search tree {30}.

43. NP - a class of problems which have the following features:

- each of them requires an answer from the set {"YES","NO"}
- a solution may be verified on the non-deterministic polynomial Turing machine.

44. NP-complete problem - a problem complete in the class NP {11, 43}.

45. Optimization problem - a problem P with a criterion of the form

$$\max\ f(x1,...,xk) \qquad (*)$$

or

$$\min f(x1,...,xk) \qquad (**)$$

where f is a function and x1,...,xk are its arguments. P requires to find the values of x1,...,xk satisfying (*) or (**).

46.Paradigm - is a conceptual pattern of something.

47. Partially-defined problem - a problem with inner nature partially or completely unknown to investigator (such a problem has a features of the "Black Box") {4, 6, 51}.

48. Polynomial algorithm - an algorithm with the complexity function restricted by some polynomial {12,21,24}.

49. Predicate - a formula with the terms as its arguments which takes only two possible values: "0" (FALSE) and "1" (TRUE).

50. Predicate calculus - a theory with the formulas representing predicates (so-called tomic formulas) or complicated formulas obtained from the atomic ones by means of the logical connectives and quantifiers {49}.

51. Problem -is a formalized mathematical structure of the form

P=<**Model, Initial_State, [Criterion], Solution, Solving_Procedure [,Proof]**>,

where:

Model is a set of relations and functional dependencies between the objects from the problem's domain;

Initial_State is a problem's specification (namely, the part which is formulated as "What is known");

Criterion represents a restriction on the **Solution**;

Solution is identified with a valid interpretation for P satisfying **Model**;

Proof is a correct reasoning which validates of the Solution {20,41,45,47}.

52. Reduceability - a possibility of the transformation of a problem X to an equivalent problem Y {9, 22, 44}.

53.Relation - a formula of predicate calculus {49,50}.

54.Search tree (Search Graph)- a tree (graph) with the nodes corresponding to the problem states and the arc corresponding to the possible moves between the states {29, 30, 42}.

55. Set characterization function - a function f(x) defined as follows:

$$\begin{cases} f(x) = 1, if\ \ x\ belongs\ \ to\ a\ given\ \ set \\ f(x) = 0, otherwise \end{cases}$$

56. Solving procedure - a procedure which finds a solution {51}.

57. Solution {see 51}.

58. State space - is a set of the problem states.

59. Term - an object of the formal theory which may be a variable, or a constant, or a functor .Terms stand for a formula's arguments {26,27}.

60. Unification - matching the arguments of the co-named predicates for example: P(a,X,f(a)) and not-P(Y,Z,Z) in order to make them indiscernible. For these two one can obtain the desired result by the substitutions:

Y=a
X=Z=f(a)

61.Universal solving strategy - an approach to solve problems by means of some universal calculus (in the sense of G.Leibnitz and D.Hilbert).

62.Weak method - is a method which does not warrant obtaining an optimal (exact) solution, e.g. heuristic method or an approximate procedure.

INDEX