A to Z of SUDOKU

Narendra Jussien

ISTE

A-Z of Sudoku

This page intentionally left blank

A-Z of Sudoku

Narendra Jussien

ISTE

First published in France in 2006 by Hermes Science/Lavoisier entitled “Précis de Sudoku”
First published in Great Britain and the United States in 2007 by ISTE Ltd

ISTE Ltd
6 Fitzroy Square
London W1T 5DX
UK

ISTE USA
4308 Patrice Road
Newport Beach, CA 92663
USA

www.iste.co.uk

Library of Congress Cataloging-in-Publication Data

Jussien, Narendra.
[Précis de sudoku. English]
A-Z of sudoku/Narendra Jussien.
p. cm.
“First published in France in 2006 by Hermes Science/Lavoisier entitled Précis de sudoku”--T.p. verso.
Includes bibliographical references and index.
ISBN-13: 978-1-84704-000-8
ISBN-10: 1-84704-000-4
1. Sudoku. I. Title. II. Title: A to Z of sudoku.
GV1507.S83J87 2007
793.74--dc22

2006037958

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 10: 1-84704-000-4
ISBN 13: 978-1-84704-000-8

Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham, Wiltshire.

Contents

Preface 9

Chapter 1. A (Mathematical) History of Sudoku 11

1.1. The rules of the game 11
1.2. A long story made short 12
1.2.1. Modern sudoku 13
1.2.2. Sudoku and medias 14
1.3. Mathematics of sudoku 16
1.3.1. Main results 16
1.3.2. Counting sudoku grids 18
1.3.3. Number of givens 20
1.4. Sudoku variants 20
1.4.1. Sudoku with additional constraints 20
1.4.2. Arithmetical variants 21

PART 1. SOLVING RULES AND TECHNIQUES 23

Chapter 2. Basic Techniques 25

2.1. Notations 25
2.2. Determining values for cells 26
2.2.1. The single position technique 26
2.2.2. The single candidate technique 28
2.3. Discarding candidates for cells 29
2.3.1. The candidate lines technique 30
2.3.2. The multiple lines technique 32

Chapter 3. Advanced Techniques 37

3.1. Pairs, triples and subsets 37

3.1.1. The naked pair technique 37
3.1.2. The naked tuples technique 38
3.2. Hidden subsets 41
3.2.1. The hidden pair technique 41
3.2.2. The naked tuple technique 42
3.3. Intrinsic properties of subset based rules 44
3.3.1. Subset-based rules duality 44
3.3.2. Some properties of region reasoning 44

Chapter 4. "Expert" Techniques 49

4.1. The XWing technique 49
4.2. The Swordfish technique 51
4.3. Trial based techniques 54
4.3.1. Disjunctive construction 54
4.3.2. *Reductio ad absurdum* 55

PART 2. COMPUTER SOFTWARE DEVELOPMENT FOR SUDOKU 57

Chapter 5. Solving Sudoku Grids 59

5.1. Logic programming 59
5.1.1. Basic principles 60
5.1.2. The PROLOG language 61
5.1.3. Logic programming and sudoku 62
5.1.4. Solving expert grids 64
5.2. Constraint programming 65
5.2.1. Basic principles 65
5.2.2. Sudoku and constraint programming 67

Chapter 6. Evaluating and Generating Grids 71

6.1. Evaluating the difficulty of a sudoku grid 71
6.1.1. A subjective issue 71
6.1.2. A pragmatic solution 73
6.1.3. The need of solving to evaluate 75
6.2. Generating sudoku grids 75
6.2.1. Top-down generation 75
6.2.2. Bottom-up generation 76
6.2.3. An open problem 76

PART 3. TRAINING 79

Chapter 7. Very Easy Sudoku Grids 81

Chapter 8. Easy Sudoku Grids 87

Chapter 9. Medium Sudoku Grids 93

Chapter 10. Difficult Sudoku Grids 107

Chapter 11. Very Difficult Sudoku Grids 121

Chapter 12. Expert Sudoku Grids 135

Appendices 151

A. Corrections 153
B. Solutions 163

Bibliography 181

Index 183

This page intentionally left blank

Preface

Sudoku is a logic puzzle that became very popular in a few months all around the world. One can find sudoku puzzles everywhere: in the subway, in trains, in buses, in classrooms, in very serious newspapers (*eg.* *Le Monde* in France or *The Times* in the UK). It is truly a game for all ages: from kids in preschool to seniors. Sudoku has a lot of intellectual stimulation virtues that promote its usage in different contexts: schools (as an initiation to mathematical logic), universities (as a support for computer science related courses), etc.

After a general introduction to the puzzle, its history and its variants, this book aims at unveiling the game:

– the first part is devoted to rules and techniques to solve sudoku grids by hand;

– the second part covers essentials of software development related to sudokus (solving, evaluating and generating grids). This is where a recent computer science branch is presented which is devoted to solving such decision support problems: constraint programming;

– finally, the third part proposes a set of grids to improve sudoku solving skills.

A set of exercises are given throughout the book. Corrections and answers are given in the appendices. Moreover, all the grids presented in the book are also corrected. In the following text, "NOTE" and "INFORMATION" sections give some complementary information that can be skipped.

Finally, a website complements the book with the source codes of all the computer programs described herein. Alongside this, a gaming interface is available on the website. It provides a solving aid based on the rules and techniques described in the book.

http://njussien.e-constraints.net/sudoku/eng-index.html

Acknowledgments

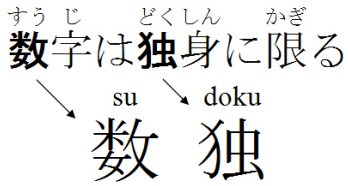
The origin of this book is the fruitful discussions the author had with François LABURTHE, Xavier LORCA, and Guillaume ROCHART.

Chapter 1

A (Mathematical) History of Sudoku

Everybody has seen them. They are everywhere. Those little grids full of digits appear in trains, in waiting rooms, in offices, etc. Even though the name of the game has a Japanese flavour, the game in its current form was invented by an American! Indeed, the first man-made grids appeared in the USA in 1979.

INFORMATION.– *The word* sudoku *is the abbreviation of a Japanese expression:* suji wa dokushin ni kagiru *(every digit must be unique). Actually, it is the name of the game when it was first published in the Japanese magazine Nikoli in 1984.* Su *means digit and* doku *means single.*



1.1. The rules of the game

The rules are simple: an 81-cell square grid is divided into 9 smaller *blocks* made up of 9 cells each. Some of the 81 cells are already filled (they are often referred to as *given*). The aim of the game is to fill all the empty cells with one digit from 1 to 9 in such a way that a digit does not appear twice in the same *row*, or in the same *column*, or in the same *block*. There is a single solution. If it is not the case, the grid is called "invalid". Figure 1.1 introduces a (valid) sudoku grid (on the left) and the way

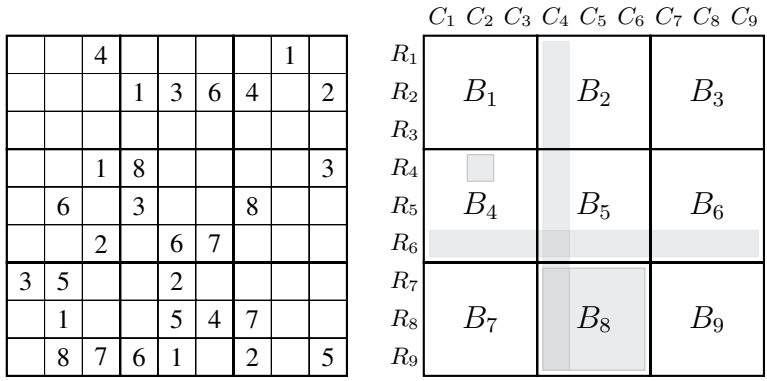


Figure 1.1. *A sudoku grid and notations to be used in the remainder of this book. For example, column C_4, row R_6, block B_8, and cell $(4, 2)$ are highlighted*

the rows, the columns, the blocks and the cells of the grid will be referred to in the remainder of this book.

Blocks B_1, B_2, and B_3 form a band (as blocks B_4, B_5, and B_6 do, as well as B_7, B_8, and B_9). Similarly, blocks B_1, B_4, and B_7 form a *stack* (as blocks B_2, B_5, and B_8 do, as well as B_3, B_6, and B_9).

INFORMATION.– *Using digits here is only a convention as in sudoku grids arithmetical relations between them are not used: they can be replaced with a set of distinctive symbols (letters, shapes, colors, etc.). Similarly, the most popular way sudoku is defined uses 81-cell grids but more general grids can be used. For example, 16-cell grids (with 4 distinct symbols – $4 \times 4 = 16$) are often used for children. Also, 256-cell grids can be found for sudoku fans involving 16 symbols ($16 \times 16 = 256$).*

1.2. A long story made short

The origin of *sudoku* can be tracked down back to 1782 with the Swiss mathematician Leonhard EULER. He introduced a famous problem on a grid: the *officer problem*.

Six regiments with six officers with distinct ranks are considered. The problem consists of placing the 36 officers into a 6×6 square. Each cell of the square contains one officer. In each line and in each column, every regiment and every rank is represented. EULER suspected that it was an unsolvable problem but could not prove it. This result was only proved in 1901 by the French mathematician Gaston TARRY.

0	1	2	3
1	2	3	0
2	3	0	1
3	0	1	2

Figure 1.2. *A latin square*

The constraint preventing the repetition of a same element in the grid is the link with sudoku. In both cases, we are faced with a particular latin square.

INFORMATION.– *A latin square is square matrix $n \times n$ filled with distincts elements and whose lines and columns contain only one instance. Figure 1.2 shows one with $n = 4$.*

1.2.1. *Modern sudoku*

At the end of the 19th century, ancestors of sudoku could be found in some French newspapers. In 1892, in the daily newspaper *Le Siècle* an 81-cell grid was given (with identified blocks) but with a 2-digit number instead of our classical digits (from 1 to 9). Figure 1.3 on page 14 reproduces such a grid.

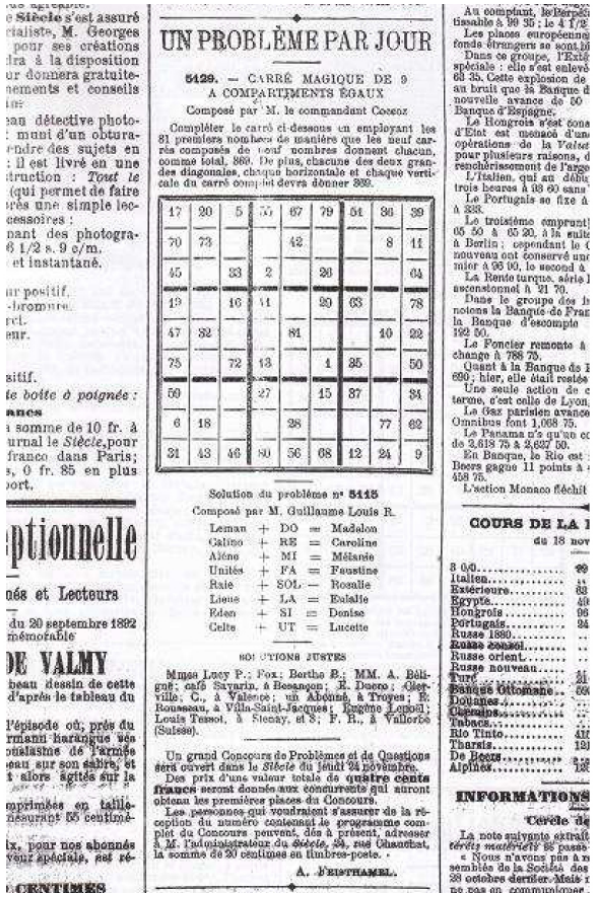
In 1895, in another newspaper, *La France* (see Figure 1.4 on page 15), a game using digits from 1 to 9 (but with no identified blocks) was given. Those weekly published games appeared in other titles in the French media but did not manage to survive WW1.

Modern sudoku was created by Howard GARNS, a retired architect and a mind puzzles fan. He introduced a new dimension in the classical latin square and presented it as a partially filled grid that was to be completed. The first grid was published in 1979 in the *Dell Pencil Puzzles and Word Games* magazine under the name *Number Place*.

INFORMATION.– *In Japan, sudoku is called* number place*. In Japanese, this is pronounced "nambaapureesu", which is often abbreviated to* nampure*!*

In April 1984, Nikoli, a Japanese editor, introduced the game in Japan under the name *sudoku*. Later on, and this is probably why the game became so popular, the number of givens was limited to 32 and published grids were made symmetrical: givens were evenly distributed around the center of the grid. Today, most prominent Japanese newspapers publish a sudoku grid daily.

In 1989, the software company LOADSTAR published the *DigiHunt* game for the Commodore 64. This was probably the first piece of software that was able to produce sudoku grids. A company still uses this name.



UN PROBLÈME PAR JOUR

5129. — CARRÉ MAGIQUE DE 9 A COMPARTIMENTS ÉGAUX

Composé par M. le commandant Coccoz

Compléter le carré ci-dessous en employant les 81 premiers nombres de manière que les neuf carrés composés de neuf nombres donnent chacun, comme total, 369. De plus, chacune des deux grandes diagonales, chaque horizontale et chaque verticale du carré complet devra donner 369.

17	20	5	76	67	79	51	36	39
70	73			42			8	11
45		33	2		20			64
19		16	11		29	63		78
47	32			81			10	22
75		72	13		1	35		50
60			27		15	37		34
6	18			28			77	62
31	43	46	80	56	68	12	24	9

Solution du problème n° 5115

Composé par M. Guillaume Louis R.

Leman + DO = Madelon
Calino + RE = Caroline
Aléne + MI = Mélanie
Unités + FA = Faustine
Raie + SOL = Rosalie
Lieue + LA = Eulalie
Eden + SI = Denise
Celte + UT = Lucette

SOLUTIONS JUSTES

Mmes Lney P.; Fox; Bertha B.; MM. A. Béligné; café Savarin, à Besançon; R. Ducro; Clerville; C., à Valence; un Abonné, à Troyes; E. Rousseau, à Villa-Saint-Jacques; Eugène Lorgeil; Louis Tessot, à Sicnay, et S. F. B., à Vallorbe (Suisse).

Un grand Concours de Problèmes et de Questions sera ouvert dans le *Siècle* du jeudi 24 novembre.

Des prix d'une valeur totale de **quatre cents francs** seront donnés aux concurrents qui auront obtenu les premières places du Concours.

Les personnes qui voudraient s'assurer de la réception du numéro contenant le programme complet du Concours peuvent, dès à présent, adresser à M. l'administrateur du *Siècle*, 14, rue Chauchat, la somme de 20 centimes en timbres-poste.

A. FEISTHAMEL.

Figure 1.3. *A pre-sudoku in the newspaper* Le Siècle *November 19, 1892*

In 2005, the *New York Post*, *USA Today* and the *San Francisco Chronicle* published their grid. In July, the game arrived in France.

1.2.2. *Sudoku and medias*

The legends says that the sudoku phenomenon was initiated by Wayne GOULD, a retired judge from New Zealand. In a Japanese bookstore, a partially filled grid caught his eye. For more than 5 years, he spent his time writing a computer software to generate such a grid (he had to learn programming from scratch). He promoted his game in *The Times*. The first grid was published on November 12, 2004. A few days later, the *Daily Mail* published its own grid under the name *Codenumber*. Since

DIVERTISSEMENTS QUOTIDIENS

N° 3879 — CARRÉ MAGIQUE DIABOLIQUE

Par M. B. Meyniel

Compléter le carré ci-dessous en employant les neuf premiers nombres chacun neuf fois de manière que les horizontales, les verticales et les deux grandes diagonales donnent toujours à l'addition le même total.

7	8	9	1	2	3	4	5	6
3				4				8
5				9				1
8				3				4
1	2	3	4	5	6	7	8	9
6				7				2
9				1				5
2				6				7
4	5	6	7	8	9	1	2	3

Ce carré devra être diabolique, c'est-à-dire que le carré restera magique si l'on place une ligne horizontale ou une colonne verticale à la suite de toutes les autres.

N° 3865 — MATHÉMATIQUES

Par M. Adolphe R.

Solution

Le marchand a vendu 27,075 vases ; le 90e jour, le dernier, il a vendu 567 vases.

Solutions justes

MM. Améthyste ; un chercheur ; Paul et Jules Duplant ; Albert Labarre ; L. Grobet ; C. Gerbaulet.

Les solutions et les envois de problèmes inédits doivent être adressés, dans la huitaine, au rédacteur soussigné.

FÉLIX ANDRÉ.

Figure 1.4. *Another pre-sudoku in* La France – *July 6, 1895*

then the *Daily Telegraph* published its grid in early 2005. At the end of 2005, Wayne GOULD was the provider for around 70 daily newspapers all around the world.

The British media realized quite early on that having a sudoku grid in a newspaper was a good thing for sales. Most national newspapers therefore started a regular publication of those grids. Indeed, sudoku is now in *The Independent*, *The Guardian*, *The Sun* (*Sun Doku*) and *The Daily Mirror*.

Sudoku's popularity is often explained by the fact that people love watching the grid filling itself. For *The Times* both easy and hard grids are appreciated. Consequently, since June 20, 2005 the two grids have been published side by side.

As early as July 2005, sudoku went to TV on British channels. The first sudoku show was aired on *Sky One*. Nine teams of nine players (including a celebrity) representing regions in Britain tried to complete a sudoku grid. Each player had a device

that allowed them to fill a digit into one of the 4 cells they were responsible for. Collaboration between players was allowed. The audience at home could play an interactive challenge.

This popularity burst made people call sudoku the "Rubik's cube of the 21st century".

The game arrived on Indian shores in June 2005 via some daily newspapers, including *The Hindu*, *Hindustan Times*, *Deccan Chronicle* and *The Asian Age*.

1.3. Mathematics of sudoku

The mathematics of sudoku is a recent field of interest and for research. The popularity of the game among mathematicians is very similar to the popularity in the media. Two main topics are addressed: complete grids study and puzzle grids. Respectively, subjects are counting possible solution grids for the game and its variants and counting the minimal number of necessary givens to obtain a valid grid.

Combinatorics and group theory are the core techniques and concepts used to address these problems. Moreover, very specific softwares are often used.

1.3.1. *Main results*

Several general results have been proved regarding sudoku, from the intrinsic complexity of the problem to its relation with other well known problems. Here are some of them.

1.3.1.1. *Complexity*

Solving a $n^2 \times n^2$ sudoku grid (consisting of $n \times n$ blocks[1]) is an NP-complete[2] problem. In other words, it is a problem for which no known efficient (both in terms of cpu time and required memory) algorithm exists in the general case.

NOTE.– *Beware! For a particular case (for a given value of n), an algorithm may exist.*

Conventional wisdom estimates that the difficulty[3] of a grid is strongly related to the number of givens. This is not the case. There exist many easy to solve grids with only 17 givens whereas really hard to solve grids can exist with more than 30 givens. Moreover, the position of the givens on the grid is also not a good indication of the difficulty. Evaluating grids will be addressed in Chapter 6.

1. Classical 82-cell grids consider $n = 3$.

2. See *Computers and intractability – a guide to the theory of NP-completeness* by GAREY and JOHNSON.

3. Difficulty – a subjective concept – is different from complexity – a mathematical concept.

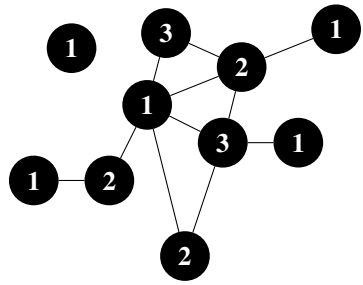


Figure 1.5. *A* 3*-colorable graph. Colors are numbered from* 1 *to* 3 *and are reported in each vertex. Two adjacent vertices always have different values*

1.3.1.2. *Sudoku and latin squares*

A valid solution of a sudoku is a latin square (see page 13). Notice that significatively fewer complete sudoku grids exist than latin squares. Indeed, there exists a supplementary constraint on blocks.

1.3.1.3. *Sudoku and graph coloring*

Completing a sudoku grid is the same as solving a graph coloring problem.

A graph is defined with a set of vertices linked by edges. A k-coloring is an assignment of a color (among k) to each vertex of the graph in such a way that two adjacent vertices (linked with one edge) have different colors. The graph coloring problem consists of determining, in a given graph, with k colors, if there exists a k-coloring of the graph (see Figure 1.5).

INFORMATION.– *Every map (representing countries) is* 4*-colorable. This is the so-called* 4*-color problem. It is a famous theorem among mathematicians because it is the first in the history of mathematics that needed a computer to be proven. Although this theorem was stated in 1852 (by a French cartographer, Francis* GUTHRIE*), it was only during the 20th century that mathematicians realized that the vast infinity of possible maps could be reduced to a finite set of representative maps. The only thing is to actually color those 1,500 maps with 4 colors. In 1976, Kenneth* APPEL *and Wolfgang* HAKEN *achieved this task way beyond human capacities. However, it was only in 1995 and 2006 that the software used to enumerate and color those maps was verified and certified (by a French team)!*

For example, consider the classical 9×9 grid ($n = 3$). Solving the problem consists of finding a 9-coloration in a specific graph that is already partially colored. Each cell represent a vertex. Edges symbolize the existing links between cells in a line, a column or a block. More formally, its vertices are labeled with their coordinates in the grid, vertices (x, y) and (x', y') are linked if and only if:

– $x = x'$ (the two vertices are on the same column);

– or, $y = y'$ (the two vertices are on the same row);

– or $x \equiv_3 x' \wedge y \equiv_3 y'$ (the two vertices are on the same block).

The grid is then solved by assigning a digit to each vertex in such a way that connected vertices always have a different value. This is a 9-coloring exactly.

1.3.2. *Counting sudoku grids*

There are several ways to count the number of existing sudokus. It depends on the definition of equivalent grids.

1.3.2.1. *Counting all solutions*

Two (solved) sudoku grids are different if they differ by the value of at least one cell among the 81. With this definition, Bertram FELGENHAUER was the first to give in 2005 the number of possible solutions:

$$6670903752021072936960 \quad \simeq \quad 6.27 \times 10^{21}$$

This number has been computed by analysing the number of possible permutations for the first band of the grid (blocks B_1, B_2, and B_3). Once the number of symmetries and equivalent classes for the partial solutions defined by this band are determined, possible completions for the other two bands are built and enumerated for each class.

INFORMATION.– *Notice that the number of* 9×9 *latin square amounts to:*

$$5524751496156892842531225600 \quad \simeq \quad 5.525 \times 10^{27}$$

Since this first computation, several others[4] have been created to count the number of possible solutions for rectangular variants of sudoku. These are given in Table 1.1.

1.3.2.2. *Counting the number of essentially distinct grids*

Two grids are *essentially* the same (equivalent) if one can be computed from the other with simple transformations. For example, the following operations always turn a valid grid into another valid grid:

– re-assigning symbols (*i.e.* performing a permutation): this can be done in $9! = 362880$ different ways;

4. Thanks to FELGENHAUER, JARVIS, PETERSEN, RUSSEL, and SILVER.

lines	columns	number
2	2	288
2	3	28,200,960 $\simeq 2.8 \times 10^7$
2	4	29,136,487,207,403,520 $\simeq 2.9 \times 10^{16}$
2	5	1,903,816,047,972,624,930,994,913,280,000 $\simeq 1.9 \times 10^{30}$
3	3	6,670,903,752,021,072,936,960 $\simeq 6.7 \times 10^{21}$
3	4	81,171,437,193,104,932,746,936,103,027,318,645,818,654,720,000 $\simeq 8.1 \times 10^{46}$
3	5	est. 3.5086×10^{84}
4	4	est. 5.9584×10^{98}
4	5	est. 3.1764×10^{175}
5	5	est. 4.3648×10^{308}

Table 1.1. *Possible solution grids for rectangular sudokus*

lines	columns	number	shown by
2	3	49	JARVIS / RUSSELL
2	4	1,673,187	RUSSELL
2	5	4,743,933,602,050,718	PETTERSEN
3	3	5,472,730,538	JARVIS / RUSSELL

Table 1.2. *Counting essentially different rectangular sudoku grids*

– permuting bands: this can be done in $3! = 6$ different ways;

– permuting lines into a band: this can be done in $3!^3 = 216$ different ways;

– permuting stacks: this can be done in $3! = 6$ different ways;

– permuting columns into a stack: this can be done in $3!^3 = 216$ different ways;

– switching lines and columns (transposition), performing a central symmetry or give a quarter turn to the grid: this can be done in 2 different ways[5].

INFORMATION.– *These operations define a symmetric relation between grids. If re-assignment is not taken into account, these operations form a sub-group from the symmetric group S_{81} of order $3!^8 \times 2 = 3{,}359{,}232$.*

Remember that, in algebra, the symmetric group of set E is the group of bijections from E to E. S_n denotes the particular case where the considered set is the set of the n first integers.

From this set of operations, the number of essentially distinct sudoku grids can be computed: 5,472,730,538.

These results has been shown by JARVIS and RUSSELL. The same reasoning can be applied to rectangular variants of sudoku. Table 1.2 gives the results.

5. Indeed, only one of these operations is sufficient because the other two can be performed with the elected one and one of the other above techniques.

1.3.3. *Number of givens*

Valid sudoku grid only have a single solution. A valid grid for which no given can be removed unless an invalid grid is obtained is called irreducible or minimal.

For classical sudoku (9×9), the smallest known number of givens that leads to a valid grid is 17. Nobody knows if a smaller number is possible. Generally, the opposite is conjectured.

INFORMATION.– *32,628 minimal essentially different 17-given grids can be found on the following website: http://www.csse.uwa.edu.au/~gordon/sudokumin.php. Figure 1.6 provides one of them.*

							1	
4								
	2							
				5		4		7
		8				3		
		1		9				
3			4			2		
	5		1					
			8		6			

Figure 1.6. *A minimal grid with only 17 givens*

1.4. Sudoku variants

It is now obvious that the most popular sudoku grid is the 9×9 grid with 3×3 blocks. However, many variants exist: for kids, 4×4 grids (with 2×2 blocks) are often used. Some 5×5 (with variable size blocks) or 6×6 (with 2×3 blocks) grids do appear at competitions. Larger grids are possible. In Japan, up to 16×16 or event 25×25 grids are available.

1.4.1. *Sudoku with additional constraints*

Main sudoku variants involve new constraints. For example, some grids have been seen with a uniqueness constraint on the diagonal. Another popular extension consists of several sudoku grids with common regions. For example, a popular version consists of 5 connected grids (4 grids sharing one block with a 5th middle grid). In Japan some puzzles involve up to 21 grids!

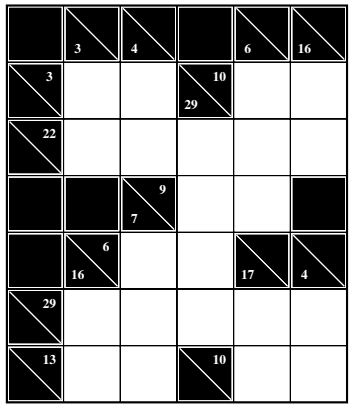


Figure 1.7. *A kakuro grid (solution in Figure 1.8)*

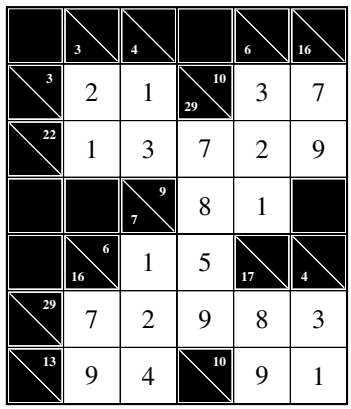


Figure 1.8. *Solution of the kakuro grid in Figure 1.7*

Alphabetical variants are often used: digits can be replaced with letters. Real words can then be hidden in the grid. Finding the word may help in solving the grid.

1.4.2. *Arithmetical variants*

The Times regularly publishes *killer* sudokus which indicate groups of cells. Each group is assigned a value that must be achieved when summing the values in the connected cells. This adds a difficulty to the grid. Finally, one cannot conclude without mentioning the *kakuro* game. This variant is often considered as a numerical adaptation of crossword puzzles. The objective is to fill lines and columns in a grid verifying sums and uniqueness of digits in them. The grid in Figure 1.7 gives an example.

This page intentionally left blank

PART 1

Solving Rules and Techniques

This page intentionally left blank

Chapter 2

Basic Techniques

Sudoku rules say that a digit must be assigned to each cell in the grid with only one restriction: a given digit cannot appear twice in a row, in a column or in a block. Obviously, one never find a sudoku grid where an single empty can be found in a row, in a column or in a block. It would be much too easy and would be of no interest. Sudoku solving requires reasoning.

In this chapter, we introduce a few basic rules and techniques. Those rules will allow us to solve any grid from *very easy* to *medium* grids[1]. The first two addressed techniques are really simple. The next ones require the use of *marks* on the grid but they are necessary for solving *medium* grids. However, our first step in this chapter will be to introduce some notations that will make it possible to properly formalize techniques and rules that we define in this book.

2.1. Notations

In this book, C, R, and B respectively refer to columns, rows and blocks in the grid. i, j, and b will be their respective index. A cell (i, j) belongs to row R_i (the ith row) and to column C_j (the jth column). $b_{i,j}$ will denote the index of the block where cell (i, j) is and $a_{i,j}$ the position of the cell in the block (from left to right and from top to bottom).

EXAMPLE.– Cell $(4, 2)$ is in block B_5. Therefore $b_{4,2} = 5$. Moreover this cell is the second cell in the block (from left to right and from top to bottom). Thus $a_{4,2} = 2$.

1. These difficulty levels are defined in Chapter 6. They are used throughout the whole book.

In this book, we will give formal rules and we will use the following notations:

– $(i, j) \leftarrow v$ states that value v must be assigned to cell (i, j);

– $(i, j) \neq v$ states that value v cannot be assigned to cell (i, j);

– $(i, j) \nleftarrow v$ states that the current value (if any) of cell (i, j) is not value v (but this value is not necessarily an impossible one for this cell).

A *region* is either a row, a column or a block in the grid. A *line* is a row or a column. A *zone* is a set of cells from the grid. Therefore, lines are regions and regions (and lines) are zones.

For a given zone Z, the set $\texttt{what}(Z)$ represents the set of possible values for at least one cell in the zone. A value is *possible* for a cell if it does not already appear in the row, the column, or the block of the considered cell. Formally:

$$\begin{aligned}\texttt{what}(Z) \quad = \quad & \{v \in 1..9 \mid \exists (i, j) \in Z, \forall j' \neq j, (i, j') \nleftarrow v \\ & \wedge \forall i' \neq i, (i', j) \nleftarrow v \\ & \wedge \forall (i', j') \neq (i, j), b_{i',j'} = b_{i,j} \rightarrow (i', j') \nleftarrow v\} \end{aligned} \qquad [2.1]$$

For a given set of values V, and for any zone Z, the set $\texttt{where}(V, Z)$ is the set of cells from Z that can take one of the values in V. Formally:

$$\texttt{where}(V, Z) = \{(i, j) \in Z \mid \texttt{what}(\{(i, j)\}) \cap V \neq \varnothing\} \qquad [2.2]$$

2.2. Determining values for cells

The first two techniques we present in this chapter enable the direct determination of the value of a cell in the grid.

2.2.1. *The single position technique*

This quite simple technique consists of scanning regions in order to find not already placed digits. A cell must be assigned a certain digit, if, because of givens and already filled cells, there exists only a single position for this digit. Formally, the following rule may be stated:

6	4		5		9	8		
				3			7	
3		8		1				4
					7			
4			9	6	5			7
			3					
8				9		4		3
	2			5				
		3	6		4		9	2

Figure 2.1. *Rule 2.1 at work*

	9		2		6		1	8
	1							
			5			9		6
6		4				8	3	
			4		2			
	8	9				2		1
2		1			9			
							6	
9	3		8		1		2	

Figure 2.2. *Rule 2.1 training*

RULE 2.1 – single position –

– **Parameters**: a region R, a digit v

– **Condition**: $\exists (i,j), \texttt{where}(\{v\}, R) = \{(i,j)\}$

– **Deduction**: $(i,j) \leftarrow v$

EXAMPLE.– Let us consider the grid in Figure 2.1. When considering column C_6, the digit 3 can only be placed in row R_8. Indeed, the 3 in cell $(2,5)$ prevents it being placed in the three first rows of column C_7. Similarly, the 3 in cell $(6,4)$ prevents its assignment in the next three rows of column C_7. Finally, the 3 in cell $(7,9)$ prevents it being placed on row R_7, as the 3 in cell $(9,3)$ does for row R_9. There exists only a single position: cell $(8,6)$.

EXERCISE 1. – On the grid in Figure 2.2, where can be placed digit 2 on column C_5?

6	4						8	
2	1							
3	8			1			6	7
7	5	8	6	2	1	3	4	9
9	2	3	4	5	7	8	1	6
4	6	1				7	5	2
1		4	2	7	8	6		5
5	7	6	1	3			2	8
8		2					7	

Figure 2.3. *Rule 2.2 at work*

EXERCISE 2. – On the grid of the previous exercise, what can be done on row R_2?

EXERCISE 3. – Using only this technique, solve the grid of exercise 1.

INFORMATION.– *Systematically applying this rule will solve all* very easy *grids.*

2.2.2. *The single candidate technique*

Consider here a cell and check the number of candidates (considering the givens in the already filled cell in the associate row, column and block). If there is only one single digit left, the value to be assigned to the cell is found!

Formally:

RULE 2.2 – single candidate –

– **Parameter**: a cell (i, j)

– **Condition**: $\exists v, \texttt{what}(\{(i, j)\} = \{v\}$

– **Deduction**: $(i, j) \leftarrow v$

EXAMPLE.– Consider the grid which is in the process of being solved in Figure 2.3. Consider cell $(1, 5)$. The only candidate digit for this cell is 9. Indeed, row R_1 already contains values 4, 6 and 8, whereas column C_5 already contains values 1, 2, 3, 5 and 7. The only remaining digit for the cell is 9. It is a single candidate.

EXERCISE 4. – What is the value of cell $(3, 4)$ in the grid in Figure 2.3?

		3	5				4	
	7	5	8	6				
	1	2		7				
			2	1				3
		6	3			5		8
						1	2	
	3			5		4		
6	8						7	
7		1						

Figure 2.4. *A medium grid*

EXERCISE 5. – At this stage of solving the grid in Figure 2.3, what are the other cells that comply with the conditions of the rule?

EXERCISE 6. – What is the resulting grid when applying this unique rule as much as possible on the same grid?

INFORMATION.– *Jointly using rules 2.1 and 2.2 solves all the* easy *grids.*

EXERCISE 7. – What is the resulting grid when applying rules 2.1 and 2.2 on the *medium* grid of Figure 2.4?

2.3. Discarding candidates for cells

The two previous rules, although really useful, are only capable of solving *easy* grids. The secret for solving other grids relies on the fact that from a certain point, the issue is not to find values to assign to cells but more to discard values for a given cell. This can become a priceless for finding the value another cell (by applying the previous rules). We will present here two such techniques.

Most human players cannot maintain such information without taking notes. We will therefore use marks on the grid. The idea is to keep track of allowed digits (candidates) for every cell of the grid. Upon deducing new information, some of them can be erased or set aside.

Figure 2.5 gives an example grid for those marks. Each cell contains its current list of candidates (no reasoning has been used, only givens are considered).

NOTE.– *The* `what` *function must be updated to take into account those marks. A marked value (removed from the candidate list) is an impossible value for the cell.*

6	4 7 9	4 5 9	8	1	2	4 5 9	3	5 9
3	1 4 8 9	1 4 8 9	4 9	5	9	6	2	7
4 5 7 9	2	1 4 5 9	3 4 7 9	3 6 9	6 7 9	1 4 5 9	8	1 5 9
9	1 3 6 9	7	5	2	8	3 9	6 9	4
2 4 5 9	3 4 6 9	2 4 5 6 9	7 9	6 9	6 7 9	3 5 7 9	1	8
2 5 8 9	1 6 8 9	1 2 5 6 8 9	1 7 9	4	3	5 7 9	5 6 7 9	2 5 6 9
7 8 9	5	8 9	6	3 8 9	4	2	7 9	1 3 9
2 4 7 8 9	4 6 7 8 9	3	2 9	8 9	1	4 5 7 9	4 5 6 7 9	5 6 9
1	4 6 9	2 4 6 9	2 3 9	7	5 9	8	4 5 6 9	3 5 6 9

Figure 2.5. *A grid with a candidate list for each cell. Cell* (i, j) *contains* $\texttt{what}(\{i, j\})$

Using this system helps to immediately identify candidate cells for rule 2.2. Notice that rule 2.1 still needs the player to scan the regions to know if it can be applied.

EXERCISE 8. – Where can rule 2.2 be applied on the grid in Figure 2.5?

EXERCISE 9. – Solve the *easy* grid of Figure 2.5.

2.3.1. *The candidate lines technique*

The next technique is the first one which is not able to directly determine the value of a cell. It only eliminates candidates for a cell. However, such an elimination may allow the application of other rules.

Consider a block B_b and a value v. Suppose that the cells able to accept this value are all localized in a unique line (row or column). It is possible to eliminate the value from all cells in that line (except of course in the considered block).

Indeed, as this value must be present in the block, placing it outside the block on the considered line will eliminate all possibilities in the block for this value. This would lead to an invalid grid.

The associated rule can be formalized in the following way (for rows):

4	5 8	9	1 6	7	1 2 6	3	5 6 8	2 6 8
3	1	7 8	5	9	2 6	4	7 8 6	2 6 8
6	7 5	2	4	8	3	1	7 5	9
5	3	4	7	6	8	9	2	1
2	9	8 6	3	1	4	7	8 6	5
7	8 6	1	2	5	9	8 6	4	3
1 8 9	4	5	1 6 8 9	3	7	2	1 6 8 9	8 6
1 8 9	7 8 6	7 8 6	1 6 8 9	2	5	8 6	3	4
1 8 9	2	3	1 6 8 9	4	1 6	5	1 6 8 9	7

Figure 2.6. *Rule 2.3 at work*

RULE 2.3 – candidate rows –

– **Parameters**: a block B_b and a value v

– **Condition**: $\exists i, \texttt{where}(\{v\}, B_b) \subseteq (R_i \cap B_b)$

– **Deduction**: $\forall (i,j) \in R_i \setminus B_b, (i,j) \neq v$

EXAMPLE.– Consider the grid in Figure 2.6. This grid contains marks that give the candidates for each cell. In block B_7, value 6 can only be on row R_8. This value is therefore forbidden in blocks B_8 and B_9 on this same row. For example, there is no 6 in cell $(8,7)$. This leaves only a single candidate (8). Rule 2.2 applies. Cell $(8,7)$ has value 8.

For columns, the rule becomes:

RULE 2.4 – candidate columns –

– **Parameters**: a block B_b and a value v

– **Condition**: $\exists j, \texttt{where}(\{v\}, B_b) \subseteq (C_j \cap B_b)$

– **Deduction**: $\forall (i,j) \in C_j \setminus B_b, (i,j) \neq v$

This technique can also be applied when, for a given line, a value is a candidate in only one of the crossed blocks. For this block, the considered value cannot be

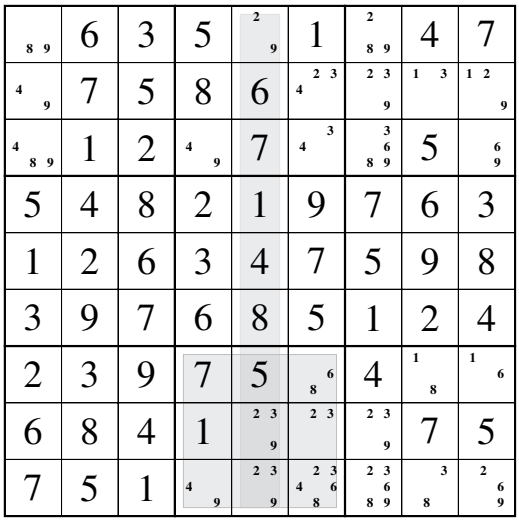


Figure 2.7. *A variant of rule 2.4*

assigned anywhere else. Figure 2.7 shows this for column C_5. Digit 3 can only be place in block B_8. Therefore, for all other cells in B_8, 3 can be eliminated from the candidate lists. This is the case for cells $(8, 6)$ and $(9, 6)$.

EXERCISE 10. – Solve the grid in Figure 2.7.

EXERCISE 11. – This variant can be applied on column C_1 in the grid in Figure 2.6. For which block and for which value?

EXERCISE 12. – Solve the grid in Figure 2.6 on page 31.

EXERCISE 13. – Consider the grid in Figure 2.8. Using rules 2.3, 2.4, and their variants, it is possible to find the value in a cell. Which one?

***Hint**: look at the gray regions.*

2.3.2. *The multiple lines technique*

The point here is to consider two contiguous blocks and a not yet placed value in those boxes. The possible cells for this value must be limited to two out of the three lines shared by those blocks. If so, the considered value must be on the third line for the third block. It can be removed from the candidate lists for the two considered lines in that third block.

Indeed, placing the value on one of these lines would eliminate all possibilities for one of the two considered blocks, leading to an invalid grid.

	6	7	5	4	2			1
2	3	4						7
						2	4	6
	4	6	7		5	1		
			1	9	6			4
		9			4		6	
4	5	8					1	
7		3	4			6	8	5
6			8	5	3	4	7	

Figure 2.8. *Training for rules 2.3, 2.4 and their variants*

Formally:

RULE 2.5 – multiple lines (for rows) –

– **Parameters**: two blocks B_{b_1} and B_{b_2} sharing a line and a value v. b_3, i_1, i_2, i_3 are the values that verify the following relationship: $B_{b_1} \cup B_{b_2} \cup B_{b_3} = R_{i_1} \cup R_{i_2} \cup R_{i_3}$.

– **Condition**: $\texttt{where}(\{v\}, B_{b_1}) \subseteq R_{i_1} \cup R_{i_2} \quad \wedge \quad \texttt{where}(\{v\}, B_{b_2}) \subseteq R_{i_1} \cup R_{i_2}$

– **Deduction**: $\forall (i,j) \in B_{b_3} \setminus R_{i_3}, (i,j) \neq v$

EXAMPLE.– Consider the grid in Figure 2.9. Value 5 in blocks B_1 and B_3 rely only in the two first rows. Therefore, it cannot be on these rows in block B_2.

NOTE.– *The rules seen here cannot always directly determine a value for a cell. However, and this is what is important here, they eliminate candidates that make applying other rules possible. This ultimately leads to finding a value for a cell. Solving a grid is therefore a matter of observation and perseverance.*

EXERCISE 14. – Give the multiple lines rule for columns (like rule 2.5).

EXERCISE 15. – Solve the grid in Figure 2.9.

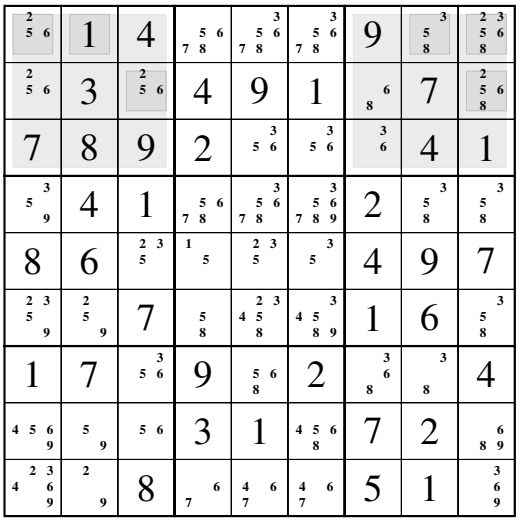


Figure 2.9. *Rule 2.5 at work*

9		3		5				6
							7	
	8	6			7	5		9
					6			2
6		5		8	3		9	
					2		6	
			3	7				8
	4	2		9	8			
	3	9						4

Figure 2.10. *A difficult grid*

INFORMATION.– *The joint use of rules from rule 2.1 to rule 2.5 can solve any* medium *sudoku grid.*

EXERCISE 16. – Solve the grid from exercise 7 on page 29.

EXERCISE 17. – What can be done in order to solve the *difficult* grid in Figure 2.10?

This page intentionally left blank

Chapter 3

Advanced Techniques

The advanced techniques presented in this chapter are meant to solve *difficult* grids.

3.1. Pairs, triples and subsets

In this section, we focus on looking for sets of particular values inside a given region.

3.1.1. *The naked pair technique*

Consider a given region. Suppose that in this region there are two cells for which the two same values are the only candidates. As such, these values can be safely removed from the candidate lists of the other cells of the region.

Indeed, assigning any of these values to another cell will leave one of the two identified cells without a potential value, which is strictly forbidden for a sudoku grid.

Formally:

RULE 3.1 – naked pair –

– **Parameter**: a region R

– **Condition**: $\exists (i_1, j_1) \in R, (i_2, j_2) \in R,$
$\texttt{what}(\{(i_1, j_1), (i_2, j_2)\}) = \{v_1, v_2\}$

– **Deduction**: $\forall (i, j) \in R \setminus \{(i_1, j_1), (i_2, j_2)\}, (i, j) \neq v_1, (i, j) \neq v_2$

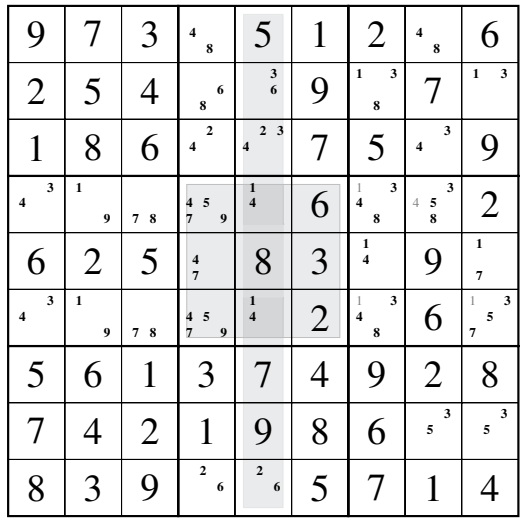


Figure 3.1. *Rule 3.1 at work*

EXAMPLE.– Consider the grid in Figure 3.1. In column C_5, cells $(4, 5)$ and $(6, 5)$ only accept values 1 and 4. Therefore, these values can be removed from the candidate lists of all the other cells in the column. Thus, the value 4 is removed from cell $(3, 5)$. The rule can also be applied on block B_5 leading to the removal of 4 from all cells in column C_4 intersecting with block B_5. It can then be deduced that cell $(5, 4)$ must be assigned with the value 7.

EXERCISE 18. – Solve the grid in Figure 3.1.

3.1.2. *The naked tuples technique*

Rule 3.1 can be rewritten to take into account k values. For example, for three values, the following formal rule can be written:

RULE 3.2 – naked triple –

– **Parameter**: a region R

– **Condition**: $\exists (i_1, j_1) \in R, (i_2, j_2) \in R, (i_3, j_3) \in R,$
$\texttt{what}(\{(i_1, j_1), (i_2, j_2), (i_3, j_3)\}) = \{v_1, v_2, v_3\}$

– **Deduction**: $\forall (i, j) \in R \setminus \{(i_1, j_1), (i_2, j_2), (i_3, j_3)\},$
$(i, j) \neq v_1, (i, j) \neq v_2, (i, j) \neq v_3$

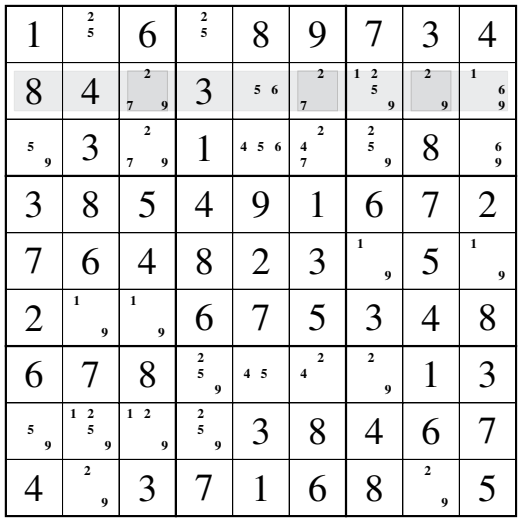


Figure 3.2. *Rule 3.2 at work*

A small issue should be considered here. Although this is not the case for the previous rule, the three values may not all be present in the three identified cells. This is the union of the candidates that form a triple over the three cells.

EXAMPLE.– On the grid of Figure 3.2, rule 3.2 can be applied on row R_2. Cells $(2,3)$, $(2,6)$, and $(2,8)$ share three candidates: 2, 7, and 9. Therefore, value 9 can be removed from cells $(2,7)$ and $(2,9)$ as well as value 2 from cell $(2,7)$.

EXERCISE 19. – Where is the triple on row R_6 in the grid of Figure 3.3?

More generally, for k values, the rule becomes:

RULE 3.3 – naked tuple –

– **Parameters**: a region R, an integer k

– **Condition**: $\exists\{(i_1,j_1),\ldots,(i_k,c_k)\} \in R^k$,
$\texttt{what}(\{(i_1,j_1),\ldots,(i_k,j_k)\}) = \{v_1,\ldots,v_k\}$

– **Deduction**: $\forall v \in \texttt{what}(\{(i_1,j_1),\ldots,(i_k,j_k)\})$,
$\forall (i,j) \in R \setminus \{(i_1,j_1),\ldots,(i_k,c_k)\}, (i,j) \neq v$

EXERCISE 20. – There is a quad in block B_9 in the grid of Figure 3.4. Can you see it?

1	2 7	2 3 7	2 3	4	9	5	8	6
2 3 8	9	5	7	2 3 6	3 6 8	1	4	2 3
2 3 8	6	4	1	5	3 8	7	3 9	2 3 9
5	3	6 9	8	1	2	4	6 9	7
7	2 8	2 9	6	3 9	4	3 8	5	1
4 6 8 9	4 8	1	5	3 7 9	3 7	3 6 8	2	3 9
4 2 3 6	1	2 3 6	2 3	8	3 5 6	9	7	3 4 5
4 3 6	4 7	8	9	3 6 7	3 5 6 7	2	1	3 4 5
2 3 6 9	5	2 3 6 7 9	4	2 3 6 7	1	3 6	3 6	8

Figure 3.3. *Looking for a triple (rule 3.2)*

8	9	1	5	7	6	2	3	4
6	3 5	3 4 5	1	2 3	2 4	9	7	8
2	3 7	3 4 7	9	3 8	4 8	5	1	6
7	8	6	2 3 4	2 3 5	2 4 5	3 4	9	1
5	1	2 9	2 3 4	6	2 4 8 9	3 4 7 8	4 8	2 3 7
3	4	2 9	7	2 5 8 9	1	6	5 8	2 5
9	2 3 5	8	2 6	1 2 5	7	1 3 4	4 5 6	3 5
4	2 5 7	2 5 7	2 6	1 2 5 9	3	1 7 8	5 6 8	5 7 9
1	6	3 5 7	8	4	5 9	3 7	2	3 5 7 9

Figure 3.4. *Looking for a quad (rule 3.3)*

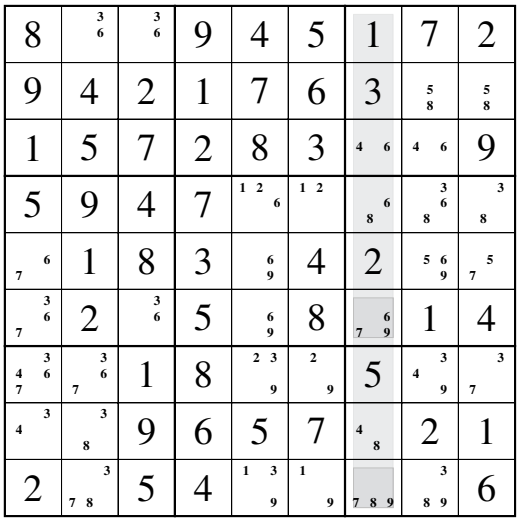


Figure 3.5. *Rule 3.4 at work*

3.2. Hidden subsets

There exist a dual set of rules (compared to the previous ones). They can be used when the subset of values that are being sought are *hidden*. Let us illustrate this duality first with hidden pairs.

3.2.1. *The hidden pair technique*

Consider a given region. If there are two values that have the same possible cell position of only two, then all other values are forbidden for these two cells.

Indeed, if any other value is assigned to one of these cells, one of the identified values will have no possible position in the region. This would not be acceptable.

Formally:

RULE 3.4 – hidden pair –

– **Parameters**: a region R, two values v_1 and v_2

– **Condition**: $\texttt{where}(\{v_1, v_2\}, R) = O_{v_1,v_2}$ and $|O_{v_1,v_2}| = 2$

– **Deduction**: $\forall v \notin \{v_1, v_2\}, \forall (i,j) \in O_{v_1,v_2}, (i,j) \neq v$

---

4 7	2	8	7 6	1 6	1 4 7 6	3	9	5
3	4 5	6	2 9	5 9	2 4 5	1	7	8
7 9	1	5 9	8	3	5 7	6	4	2
1	3	7	5	2	9	8	6	4
4 9	4 5	2 5 9	3 6 7	8	3 6 7	2 9	1	3 7 9
6	8	2 9	1	4	3 7	5	2 3	3 7 9
5	6	1	2 3 9	7	8	4	2 3	3 9
2	9	3	4	1 5	1 5	7	8	6
8	7	4	2 3 6 9	6 9	2 3 6	2 9	5	1

Figure 3.6. *Rule 3.5 at work*

EXAMPLE.– In the grid of Figure 3.5, values 7 and 9 on column C_7 are candidates in only two cells. All other values may be removed from those two cells.

NOTE.– *Inferring such information is useful because other rules may apply.*

EXERCISE 21. – Which other rule (from this chapter) can be applied here with exactly the same result?

3.2.2. *The naked tuple technique*

As for rule 3.1, the previous rule may be generalized to k values. Therefore:

RULE 3.5 – hidden tuple –

– **Parameter**: a region R, a set V of k values

– **Condition**: $\texttt{where}(V, R) = O_V$ and $|O_V| = k$

– **Deduction**: $\forall v \notin V, \forall (i,j) \in O_V, (i,j) \neq v$

NOTE.– *Be careful, because the same issue as above arises: values may not all be present in the considered cells.*

EXAMPLE.– On row R_5 in the grid in Figure 3.6, values 3, 6, and 7 are only possible in three cells: $(5,4)$, $(5,6)$, and $(5,9)$. Thus, the rule applies and value 9 can be removed from cell $(5,9)$.

9		3		5				6
							7	
	8	6			7	5		9
					6			2
6		5		8	3		9	
					2		6	
			3	7				8
	4	2		9	8			
	3	9						4

Figure 3.7. *A difficult grid*

6	4				9		8	
8			4			6		
	9		3		8		5	
	7	9		8	2			
4				7	6			
						8		
7	2	8				5	9	
9			8	5		7		6
	6			9				

Figure 3.8. *A very difficult grid*

INFORMATION.– *The joint application of the rules from rule 2.1 up to rule 3.3 (when only considering $k \leq 3$) can solve any difficult grids.*

EXERCISE 22. – Solve the difficult grid in Figure 3.7.

EXERCISE 23. – Use the learnt techniques as much as possible on the grid in Figure 3.8. What is the resulting grid?

In the next chapter we will solve very difficult grids, but, before that, it is worth having a closer look at the rules presented in this chapter.

3.3. Intrinsic properties of subset based rules

The two sets of rules that we have presented are strongly related. We have seen this when answering exercise 21. We will now explicitly exhibit this relation, and we will also show that all these rules are subsumed by a more powerful and general rule.

3.3.1. *Subset-based rules duality*

Let R be a region with p non-assigned cells. It can easily be shown that rule 3.3 applied to region R for n values has the same result as rule 3.5 applied to region R and $p - n$ values.

Indeed, these two rules define a partitioning of the region into:

– a set E of $9 - p$ assigned cells;

– a set F of n cells for which globally only n candidates are available;

– a complement set G of $p - n$ cells which globally contain $p - n$ values that can be assigned on cells other than in G.

Rules 3.3 and 3.5 both forbid cells in G to receive a value shared by cells in F.

EXAMPLE.– In the grid in Figure 3.1 on page 38, consider column C_5. Using rule 3.5 on values 2, 3 and 6 (which have only three candidate rows R_2, R_3 and R_9) leads to the removal of value 4 from cell $(3, 5)$. This is exactly the same conclusion as considering rule 3.3 on rows R_4 and R_6 which only have two possible values: 1 and 4.

EXERCISE 24. –

What partitioning can be identified when considering row R_5 in the grid of Figure 3.6 on page 42?

3.3.2. *Some properties of region reasoning*

Reasoning on a given region of a sudoku grid leads to a common situation in graph theory: the *maximum matching problem*. On one side, the cells of the region are considered. On the other side, the digits from 1 to 9 are considered. What is needed is

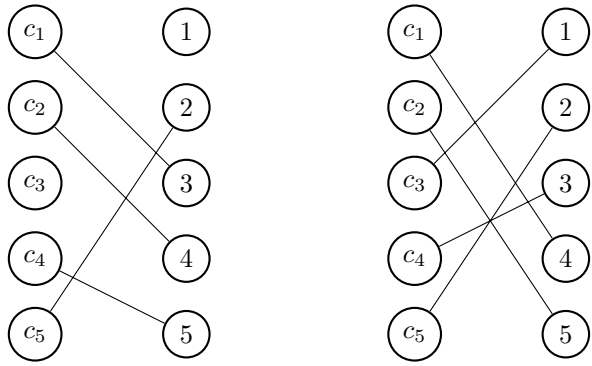


Figure 3.9. *Two example matchings. The matching on the left is not a maximal matching (cell c_3 has no match). However, the matching on the right is*

to assign to each cell a unique digit in such a way that a value is not assigned to two different cells. This is a *matching*. It is a *maximum* matching because all cells must have a value.

EXAMPLE.– Consider five cells $(c_1, \ldots, c_5)$ and five candidates $(1, \ldots, 5)$. Figure 3.9 gives two example matchings. Such a matching can be represented as a graph in which left vertices are the cells and right vertices are the candidates. A link (an edge) between a cell and a candidate represents the fact that the value is assigned to the cell. In these examples, we can clearly that no two edges have a vertex in common.

When considering a sudoku grid, a value cannot be assigned to all the cells: some already have one (the givens), others are restricted to their candidate list, etc. This information has to be taken into account. Therefore, a specific graph is designed, in which a maximal matching is to be found: a cell is linked to a digit if the digit is a valid candidate for this cell.

EXAMPLE.– Let us get back to our five cells and five values. Figure 3.10 gives an example graph. Here, we have: $\texttt{what}(\{c_1\}) = \{1, 2, 3\}$, $\texttt{what}(\{c_2\}) = \texttt{what}(\{c_3\}) = \{2, 3\}$, $\texttt{what}(\{c_4\}) = \{1, 2, 4, 5\}$, and $\texttt{what}(\{c_5\}) = \{3, 4, 5\}$. A maximal matching is sought using only the edges of this graph. Figure 3.11 gives two such matchings (the edges in the matching are in bold).

In the specific context of sudoku, what is looked for is not a solution to this problem. Indeed, such a solution may actually not fit with the other regions of the grid.

EXAMPLE.– This can be clearly seen on the previous example (see Figure 3.11). The solution for a sudoku grid is unique: there is no way to tell which of the two matchings will lead to a solution.

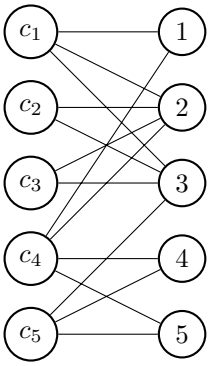


Figure 3.10. *A sudoku-like situation. A maximal matching is needed in this graph*

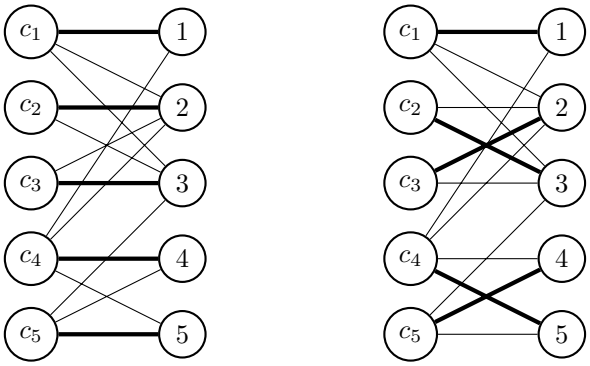


Figure 3.11. *A sudoku-like situation: two example matchings*

Indeed, what is needed are:

– the mandatory assignments (not counting givens): are there any cell/value couples that appear in all the solutions of this problem?

– the forbidden assignments: are there any cell/value couples that never appear in a solution of this problem?

EXAMPLE.– Consider Figure 3.10. The following points can be identified:

– one mandatory assignment: c_1 is bound to have value 1. Indeed, the other candidates are 2 and 3 but these values are the only possible ones for a pair of cells (c_2 and c_3). Therefore, rule 3.1 (the naked pair rule) can be applied and 1 becomes the only valid candidate for c_1 (rule 2.2);

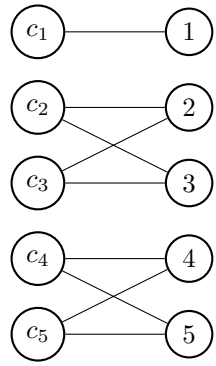


Figure 3.12. *Reasoning about maximum matchings*

– several forbidden assignments: c_4 cannot take value 1 (it has been assigned to c_1) and cannot take value 2 for the same reason as before. This is also the case for cell c_5 and value 3.

Thus, c_1 must be assigned to value 1, whereas 2 and 3 are shared by c_2 and c_3, and 4 and 5 are shared by c_4 and c_5. Indeed, making any other combination assignment will make it impossible to reach a maximal matching. The graph of Figure 3.10 becomes that in Figure 3.12: edges that correspond to values removed from the candidate lists were removed from the graph.

From an algorithmic point of view, identifying mandatory assignments is quite easy. This is exactly what rules 2.1 and 2.2 do. A simple (i.e., there exists an efficient algorithm to achieve the calculation) way to answer the second question (forbidden assignments) consists of identifying set of auto-sufficient and independent elements (cells and/or values). This is partly what rules 3.1 to 3.5 do with the partition that is provided. Any cell/value assignment *linking* any two of these sets will never appear in a solution.

EXAMPLE.– In Figure 3.10 (page 46), the 4 vertices c_2, c_3, 2, and 3 form an auto-sufficient set[1]. Trying to give another value to c_2 or c_3 will never lead to a maximal matching. The same applies if one tries to assign value 2 or 3 to another cell.

EXERCISE 25. – What are the other auto-sufficient sets in Figure 3.10?

1. The technical name is *strongly connected component* in an oriented graph where the edges *in* the matching are oriented from left to right and the edges *not in* the matching are oriented from right to left.

The rules presented here are all a particular case of this more general reasoning. There are very efficient algorithms that can be used to solve this problem. They can deduce several pieces of information at the same time. Unfortunately, using such a reasoning by hand is a quite difficult and tedious task. However, these algorithms are very useful when considering developing software to solve sudoku grids. This will be the topic of Chapter 5.

INFORMATION.– *The results presented here are part of a well-known tool in the constraint programming community: the* alldifferent *constraint. For more information, see the chapter* Global constraints and filtering algorithms *written by Jean-Charles* RÉGIN *in the book* Constraints and integer programming combined *edited by Michela* MILANO *and published by Kluwer in 2003, which gives more detail on involved algorithms.*

Chapter 4

"Expert" Techniques

In this chapter, we present the techniques that can be used to solve *very difficult* and *expert* grids.

4.1. The XWing technique

As for the previous chapter, rules and techniques in this chapter are devoted to determine impossible candidates for cells rather than finding the value to be assigned on a specific cell.

The *XWing* technique can be applied when there are two lines for which only two cells are possible for a given value. If those four cells are in only two orthogonal lines, then all other cells in those regions will never get assigned to this value.

EXAMPLE.– Consider the grid of Figure 4.1. In column C_5, value 7 is only possible on rows R_2 and R_4. This is also the case for column C_9. This is indeed an *XWing* configuration: C_5 and C_9 are two lines for which value 7 is a candidate in only two cells. Moreover, these four cells are placed on to only two rows (R_2 and R_4). Value 7 can be safely removed from the candidate lists of all other cells in these rows. Indeed, two cases are to be considered:

– for column C_5, if the 7 is assigned to row R_2, then for column C_9 the 7 must be on row R_4;

– for column C_5, if the 7 is assigned to row R_4, then for column C_9, the 7 must be on row R_2.

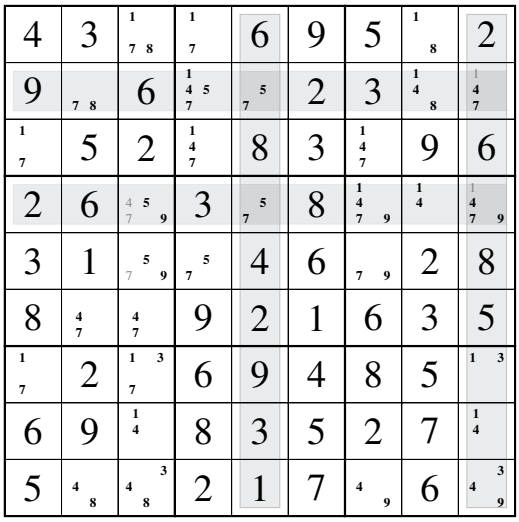


Figure 4.1. *Rule 4.1 at work. In gray, candidates that have been eliminated by other techniques*

In all possible configurations, there is a 7 in columns C_5 and C_9 on rows R_2 and R_4 (although the precise position is not known). Therefore, the 7 cannot be in any other cells in those lines. It can be safely removed from the candidate lists of those cells.

Formally:

RULE 4.1 – XWing (for columns) –

– **Parameter**: a value v

– **Condition**: $\exists i_1, i_2, j_1, j_2, \texttt{where}(\{v\}, C_{j_1}) = C_{j_1} \cap (R_{i_1} \cup R_{i_2})$
$\wedge \quad \texttt{where}(\{v\}, C_{j_2}) = C_{j_2} \cap (R_{i_1} \cup R_{i_2})$

– **Deduction**: $\forall j, j \neq j_1, j \neq j_2, (i_1, j) \neq v \wedge (i_2, j) \neq v$

INFORMATION.– *The name of this technique comes from the X that the four distinguished cells form on the grid. It is also a reference to the rebel fighters in the movie "Star Wars".*

EXERCISE 26. – Find an *XWing* configuration on the grid of Figure 4.2.

Hint: *have a look at columns* C_3 *and* C_8.

EXERCISE 27. – Solve the grid of the previous exercise.

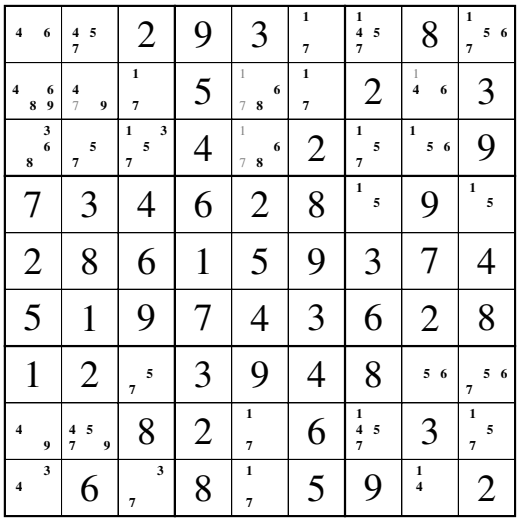


Figure 4.2. *Looking for an* XWing *configuration. In gray, candidates that have been eliminated by other techniques*

4.2. The Swordfish technique

The *XWing* technique can easily be generalized. Instead of looking for two lines on which only two cells make take a given value, it is possible to look for several lines. However, there must be a *path* from cell to cell alternatively going through a row and a column[1]. When only considering two lines, the *XWing* configuration (an X-figure) is obtained. When considering three lines, an L-figure is obtained.

EXAMPLE.– Consider the grid in Figure 4.3. Value 1 is a candidate for only two cells on rows R_4, R_5, and R_9. Moreover, starting from cell $(4, 7)$ and taking lines R_4, C_4, R_5, C_3, R_9, and finally C_7, all the identified cells can be visited. Thus, candidate 1 can safely be removed from all other cells in columns C_3, C_4, and C_7. Indeed, as for the *XWing* technique, it is quite obvious that value 1 must only be on the identified lines and nowhere else for these three columns.

INFORMATION.– *The name of the technique comes from the figure displayed when highlighting the selected cells: a swordfish.*

Formally:

1. The same reasoning applies when considering blocks but it is much more difficult to see by hand.

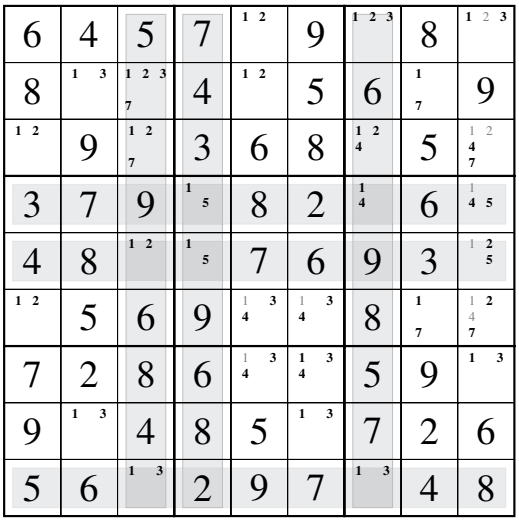


Figure 4.3. *Rule 4.2 at work*

RULE 4.2 – Swordfish (for lines) –

– **Parameter**: a value v

– **Condition**: $\exists i_1, ..., i_k, j_1, ..., j_k$ such that:
- $\texttt{where}(\{v\}, R_{i_1}) = R_{i_1} \cap (C_{j_1} \cup C_{j_2})$
- $\texttt{where}(\{v\}, R_{i_2}) = R_{i_2} \cap (C_{j_2} \cup C_{j_3})$
- ...
- $\texttt{where}(\{v\}, R_{i_{k-1}}) = R_{i_{k-1}} \cap (C_{j_{k-1}} \cup C_{j_k})$
- $\texttt{where}(\{v\}, R_{i_k}) = R_{i_k} \cap (C_{j_k} \cup C_{j_1})$

– **Deduction**: $\forall i, i \notin \{i_1, \ldots, i_k\}, \forall j \in \{j_1, \ldots, j_k\}, (i, j) \neq v$

EXERCISE 28. – Find a Swordfish configuration (for columns) on the grid in Figure 4.4.

Hint: *look for candidate* 9 *on columns* C_1, C_5, *and* C_6.

EXERCISE 29. – Solve the grid in the previous exercise.

INFORMATION.– *Using rule 4.2 (and its particular instance: rule 4.1) makes it possible to solve any very difficult sudoku grid. Grids that cannot be solved using those rules are called expert grids.*

EXERCISE 30. – Try to solve the grid in Figure 4.5 using only the rules described in this book.

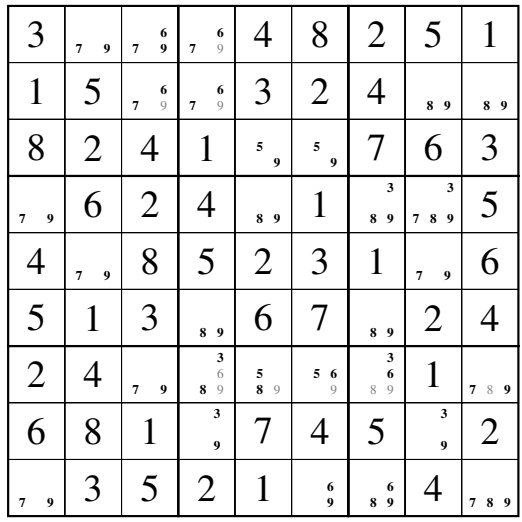


Figure 4.4. *Looking for a Swordfish configuration (rule 4.2)*

					4		7	8
	5		9		6	4		
		9			8	6	2	
	9	6				3		
		2		6		7		
				9			5	
		4		7				
			6		3			
		7			2		9	

Figure 4.5. *An example expert grid*

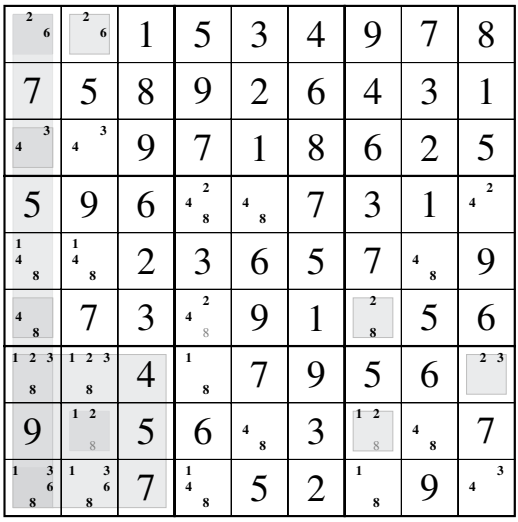


Figure 4.6. *Disjunctive construction at work*

4.3. Trial based techniques

The remaining techniques are designed to solve expert grids when none of the previous rules can be applied.

4.3.1. *Disjunctive construction*

The principle is easy to understand: choose a cell for which there remains only a few possible candidates and try the different values alternatively. If, for every choice, a same value is assigned to a same cell, this cell should definitely be assigned this value in order to reach the solution. Conversely, nothing can be deduced and then another cell or the next technique in this chapter should be used.

INFORMATION.– *This is a constructive trial because the tree of all possibilities is built and one attempts to identify common branches.*

EXAMPLE.– Consider the grid in Figure 4.6. None of the rules presented in this book is applicable. Consider cell $(8, 2)$. Only two cells are possible: 1 and 2. There are two cases:

– assigning value 2 to cell $(8, 2)$ leads to the following deductions:
- $(1, 2) \leftarrow 6$ (rule 2.2)
- $(1, 1) \leftarrow 1$ (rule 2.2)
- $(9, 1) \leftarrow 6$ (rule 2.1)

– assigning value 1 to cell $(8, 2)$ erases this value from all the candidate lists of the other cells of the corresponding row, column and block. Another important deduction can be made: assigning value 2 to cell $(8, 7) \leftarrow 2$ (rule 2.2). This leads to:

- $(7, 9) \leftarrow 3$ (rule 2.2)
- $(9, 1) \neq 8$ (rule 3.1 for block B_7 on values 2 and 8)

but also to:

- $(6, 7) \leftarrow 8$ (rule 2.2)
- $(6, 1) \leftarrow 4$ (rule 2.2)
- $(3, 1) \leftarrow 3$ (rule 2.2)
- $(9, 1) \neq 3$

There remains only one candidate in $(9, 1)$: value 6.

Consequently, trying all the candidates to cell $(8, 2)$ leads to value 6 being assigned to cell $(9, 1)$. This cell can therefore be assigned safely.

EXERCISE 31. – Try to continue to solve the grid (no longer using the disjunctive construction). What is the resulting grid?

4.3.2. *Reductio ad absurdum*

The idea consists of choosing an unassigned cell and considering one of its candidates. If assigning this value to the cell leads to a contradiction (no more candidates for one cell, no more possible cells for a value, etc.), the tested value can be safely removed from the candidate list.

NOTE.– *This is really the only deduction that can be done. Indeed, not encountering a contradiction does not mean that the selected value is a good one: the contradiction may happen later.*

EXAMPLE.– Consider the grid in Figure 4.7. No rule can be applied. Thus, consider cell $(8, 2)$ and candidate 2. Suppose that this value is assigned to the cell. Then, value 1 must be assigned to cell $(8, 7)$ and therefore value 2 to cell $(6, 7)$ (because of rule 2.1 on column C_7). But, then, the only candidate in cell $(4, 9)$ is a 4. This leads to assigning a 3 to cell $(9, 9)$ which implies a 4 in cell $(9, 4)$ (rule 2.1 for row R_9). There is then no more candidates in cell $(6, 4)$. Therefore, cell $(8, 2)$ cannot be assigned value 2. The right value to be assigned is therefore 1.

Many variants of this rule exist. For example, the *Nishio* technique focuses on a single value for the whole grid.

INFORMATION.– *With all the rules and techniques described in this chapter and the previous ones, it is now possible to solve any expert grid. Even sudoku variants may be solved. It is now your turn!*

2	6	1	5	3	4	9	7	8
7	5	8	9	2	6	4	3	1
4 3	4 3	9	7	1	8	6	2	5
5	9	6	4 2 8	4 8	7	3	1	4 2
1 4 8	1 4 8	2	3	6	5	7	4 8	9
4 8	7	3	4 2 8	9	1	2 8	5	6
1 3 8	1 2 3 8	4	1 8	7	9	5	6	2 3
9	1 2 8	5	6	4 8	3	1 2 8	4 8	7
6	1 3 8	7	1 4 8	5	2	1 8	9	4 3

Figure 4.7. *Reductio ad absurdum at work*

EXERCISE 32. – Solve the grid in Figure 4.7.

INFORMATION.– *Rules and techniques that are presented in this book and that perform some reasoning on a given region are all applicable in the specific context of kakuro (see page 21). Obviously, other techniques may be considered in order to take into account the information regarding the sum of the value on a given region of such a grid.*

PART 2

Computer Software Development for Sudoku

This page intentionally left blank

Chapter 5

Solving Sudoku Grids

As shown previously, the reader now knows that sudoku grids can be solved by systematically and iteratively applying a set of rules (some simple, others not). Most grids can be solved using only these rules. However, as we saw in the previous chapter, *expert* grids require some trial-based techniques to be solved: arbitrary choices are made; these choices must be undoable (as is the case in the techniques presented in section 4.3).

In designing a computer application to solve sudoku grids, implementing the rules of this book is quite easy. However, implementing backtracking[1] can be tricky.

Hopefully, a full branch of computer science coming from formal logics, the so-called *logic programming* paradigm, fully integrates this mechanism. We shall see later how to use logic programming to write a sudoku solver.

Our next subject will be a recent extension of logic programming, constraint programming which is so powerful that it enables us to write in very few lines a sudoku solver that can be used for any kind of grid.

5.1. Logic programming

PROLOG, the first logic programming language[2], was born in 1972 in Marseilles, France thanks to Alain COLMERAUER and Philippe ROUSSEL. The objective was to

1. This is what going back to a previous situation after a (bad) choice has been made is called.

2. *The Art of Prolog* by Leon STERLING and Ehud SHAPIRO is a good reference book for a complete and precise description of PROLOG.

define a programming language based on the expressivity of formal logics to solve a problem instead of being forced to given a step-by-step algorithm. Thus, a *declarative* language was desired instead of an *imperative* one.

PROLOG is widely used in artificial intelligence software and for natural language processing.

INFORMATION.– *Implementing a* PROLOG *interpreter is a fascinating exercise.* The implementation of Prolog *by Patrice* BOIZUMAULT *is a very interesting read.*

5.1.1. *Basic principles*

Logic programming is based upon results from the beginning of the 19th century due to the French mathematician Jacques HERBRAND. In 1965, Alan ROBINSON made them operational. The logics in question are First Order logics (variables and quantifiers – $\forall$ and $\exists$). The principle of PROLOG is to prove theorems from a set of facts and rules.

For example, if we consider the rule "all humans are mortals", knowing that "Socrates is a human" makes it possible to prove the fact that "Socrates is mortal".

Using First Order logics, the rule can written:

$$\forall x, \qquad \mathrm{human}(x) \quad \longrightarrow \quad \mathrm{mortal}(x) \qquad [5.1]$$

In the previous equation, one can find the predicate[3] $\mathrm{human}(x)$ to express the fact that "x is human"and $\mathrm{mortal}(x)$ for "x is mortal".

The fact itself can be written:

$$\exists y, \qquad \mathrm{human}(y) \qquad [5.2]$$

Here, y represents Socrates.

What is being proved is:

$$\exists z, \qquad \mathrm{mortal}(z) \qquad [5.3]$$

This result can be easily proved using the so-called HERBRAND's theorem. Actually, the proof also shows that there is identity between y (from equation 5.2) and z (from equation 5.3).

3. A predicate is a relation that is verified (or not) by its parameters. It has essentially a boolean value.

5.1.2. *The* PROLOG *language*

One of the main advantages of PROLOG is that it is only necessary to write in the language, the rule, the fact and the proposition to be proven. The proof itself is entirely automatic.

Here, we write the rule the following:

```
mortal(X) :- human(X).
```

A PROLOG variable is an expression starting with a capital letter. The ":-"part should read "if". Implications are thus written in the opposite fashion to the traditional way.

The fact is written:

```
human(socrates).
```

Finally, interacting with PROLOG consists of asking questions. Each one is called a query. Here, one is looking for a variable `Y` which can verify the mortal predicate:

```
?- mortal(Y).
```

The PROLOG system computes a proof and can say that in order to verify the query, `Y` should take the value `socrates`. One gets as expected: "Socrates is mortal". The instantaneous answer is:

```
Y = socrates
```

This illustrates one of the fundamental properties of PROLOG: declarativity. In simpler terms, it expresses the capacity of the language to only describe the properties of a solution that is sought instead of specifying the complete set of operations to be done to actually calculate the solution.

Another important property of PROLOG is the trial-based procedure to solve problems. When failures are encountered, the system performs backtracking by itself. Backtracking amounts to getting back to a previously recorded situation to take another decision than the one that led to the failure.

5.1.3. *Logic programming and sudoku*

PROLOG[4] is a good candidate to solve sudoku grids.

In the following, we will give the PROLOG code to solve 4×4 sudoku grids (these are often used for children). Its generalization to the "classical"sudoku is quite easy.

Using PROLOG declarativity, it is necessary to specify the properties to be verified by a valid sudoku solution grid, to provide the given and finally to ask the PROLOG system to give a value to the variables.

We consider here one variable per cell. Its value will be the digit to write in the cell in the solution grid (which is necessarily unique).

A valid solution obeys two principles: uniqueness of each digit in a given region and a restriction to digits from 1 to 4. The latter is the easier to take into account in PROLOG:

```
restriction(X) :- member(X, [1,2,3,4]).
```

Indeed, `X` verifies the restriction property if `X` is an element of the list of the four first integers[5].

Uniqueness is not more complex to implement:

```
unique([]).
unique([C|Cs]) :-
  not(member(C,Cs)),
  unique(Cs).
```

The uniqueness property is verified for each empty region (`[]`). Moreover, if a region is composed of a cell `C` and a set of other cells `Cs` (this is what is meant by the `[C|Cs]` notation), then:

– on the one hand, the value of cell `C` should not be present among the other values in the region;

4. There are many freeware PROLOG interpreters. For example, `SWI prolog` is available under GPL license for linux and Windows; `GNUprolog` is available for a wide variety of platforms under a GNU licence.

5. `member` is predefined predicate in PROLOG.

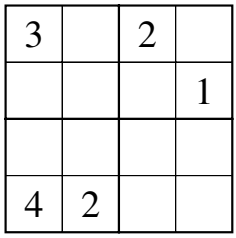


Figure 5.1. *A* 4×4 *sudoku grid*

– on the other hand, the uniqueness property should be globally verified for the remainder of the region.

This is another important property of logic programming: recursivity (a predicate calling itself) which is the computer science counterpart of mathematical recurrence.

Finally, the grid is represented with the list of its 16 cells. The following program is obtained:

```
sudoku([C11, C12, C13, C14, C21, C22, C23, C24,
        C31, C32, C33, C34, C41, C42, C43, C44]) :-
 restriction(C11), restriction(C12), ..., restriction(C44),
 % for the lines
 unique([C11,C12,C13,C14]), ..., unique([C41,C42,C43,C44]),
 % for the columns
 unique([C11,C21,C31,C41]), ..., unique([C14,C24,C34,C44]),
 % for the blocks
 unique([C11,C12,C21,C22]), ..., unique([C33,C34,C43,C44]).
```

If the aim is to solve a grid such as that in Figure 5.1, then the corresponding PROLOG query would be:

```
?-  sudoku([  3, C12,   2, C14, C21 , C22, C23,   1,
            C31, C32, C33, C34,   4,    2, C43, C44]).
```

The query leads to the following answer:

```
C12 = 1, C14 = 4,
C21 = 2, C22 = 4, C23 = 3,
C31 = 1, C32 = 3, C33 = 4, C34 = 2,
C43 = 1, C44 = 3
```

3	1	2	4
2	4	3	1
1	3	4	2
4	2	1	3

Figure 5.2. *A 4×4 sudoku grid solved*

which is exactly the solution that we are looking for. The grid in Figure 5.2 shows it.

NOTE.– *A few seconds are needed to solve this grid.*

5.1.4. *Solving expert grids*

The previous lines of PROLOG represent a program that is able to handle any sudoku grid. The way the PROLOG interpreter works is by using a trial-based technique. Indeed, using the `restriction` predicate leads to the complete enumeration of the possible solutions until the good one is found.

NOTE.– *The number of such possible solutions can be quite huge. This method of solving sudoku grids can therefore take a very long time. This is why solving the grid in Figure 5.1 is not instantaneous.*

One may regret that the human *expertise* (as we saw in the first part of this book) to solve sudoku is completely set aside by such a *brute force* technique.

As we saw in section 1.3.1.1 (page 16), the general problem of solving a sudoku grid is a difficult problem, so this technique is bound to encounter computation time difficulties. It works for 4×4 grids, but it becomes unfeasible for traditional 9×9 grids.

The question now is to know if there is a way to use this accumulated expertise but without an explicit implementation[6]. Actually, there is one extension of logic programming that enables the use of such information to solve problems. It is called *contraint programming*.

6. Indeed, the set of rules of the first part of the book could have been written in PROLOG. Notice that the readability of the obtained program would not have been comparable.

5.2. Constraint programming

Fast increasing computing power in the 1960s led to a wealth of works being produced on problem solving, at the root of operational research, numerical analysis, symbolic computing, scientific computing, and a large part of artificial intelligence and programming languages. Constraint programming is a discipline that gathers, interbreeds, and unifies ideas shared by all these domains to tackle decision support problems.

This technology has become in 15 years a leading technique in modeling and automatically solving real-life combinatorial problems. It has been successfully used in various fields such as production planning and scheduling, gate assignment in airports, frequency allocation, diagnosis and testing, molecular biology, robotics and industrial process control.

INFORMATION.– *The Association for Constraint Programming (*ACP*) aims at promoting constraint programming in every aspect of the scientific world by encouraging its theoretical and practical developments, its teaching in the academic institutions, its adoption in the industrial world, and its use in the application fields.* `http://slash.math.unipd.it/acp/index.html`

5.2.1. *Basic principles*

Constraint programming manipulates two simple concepts:

– some (mathematical or logical) variables. Each variable takes its value in a given domain;

– some constraints: relations to be verified by a set of variables.

What is it sought is called a solution: a value for each variable (from its domain) that simultaneously verify all the constraints of the problem. A constraint solver is meant to look for such a solution. End-users of constraint programming only need to enunciate the variables and the constraints of their problem.

Just before showing how to use constraint programming to solve sudoku grids, it is worth having a look at the internal behavior of a constraint solver as it is strongly related to the way sudoku grids are solved by human players:

– one does not directly look for values for cells but more to values that cannot be taken by variables in any solution (as in sudoku where candidates to eliminate are looked for): this is called domain reduction;

– each constraint is handled locally with a double aim: to ensure that choices already made are consistent (this is called a satisfiability test) and to eliminate impossible values as much as possible (this is called filtering). In sudoku solving, these behaviors are modeled in the rules presented earlier (see Chapter 3);

– finally, constraints communicate through variables: when the current domain of a variable has changed, all constraints on the considered variable are *awoken* in order to compute new deduction. This is called propagation. In sudoku solving, this is the same: as soon as some deduction has been made, all relevant rules are tested.

INFORMATION.– *This is the notion of constraint that shows the difference between logic programming and constraint programming. In the latter, constraints are used to perform filtering, whereas in logic programming their use is for a posteriori solution verification.*

A constraint solver is a software tool that implements those principles.

EXAMPLE.– Consider three variables x, y, and z with a given discrete domain (the same for the three variables): $[1, 2, 3]$. The objective is to find a value for these variables such that $x > y$ and $y > z$. Here is how a constraint solver would automatically proceed:

1) Consider $x > y$: x cannot be assigned to value 1 because y would not have any possible value left. Similarly, y cannot be assigned to value 3 because there would be no value left for x. Thus, domains for x and y become (respectively) $[2, 3]$ and $[1, 2]$.

2) Consider $y > z$: y cannot be assigned to value 1 because of z (which would be left with an empty domain); z cannot be assigned to value 2 or to value 3 because the domain of y would be empty. Thus, we have shown that y must be assigned value 2 and that z must be assigned value 1.

3) As the domain of variable y has changed, constraint $x > y$ should be reconsidered. As y is equal to 2, it can be shown that x is bound to have value 3.

The three variables are now all assigned. The problem is solved.

Every constraint solver comes out with a predefined library of constraints. This large set of filtering techniques is an invaluable tool to be used to model and solve quite a large number of types of problems.

INFORMATION.– *Being able to characterize the filtering that is achieved by a constraint is something which is important for constraint solvers. Indeed, constraints may be used a posteriori for checking solutions, but their real interest lies in their use for reducing domains. For example, a common characterization (called arc-consistency) can be stated quite easily:*

> *A constraint is arc-consistent as soon as for each variable and for each value in the domain of that variable there exists a value assignment (compatible with the domains) for the whole set of the other variables.*

It is not always easy to ensure this property. An interesting entry point for this problem is a paper written by Romuald DEBRUYNE *and Christian* BESSIÈRE *in the Journal of Artificial Intelligence Research entitled* Domain filtering consistencies.

Obviously, this constraint propagation process does not always converge towards a solution (a value for each variable). It is then necessary to switch to a so-called enumeration phase. This phase essentially consists of picking a value for a variable and propagating this commitment. The process is iterated until a solution is found or a contradiction is identified. In the former case, the search is finished. In the latter, it is necessary to reconsider the last choice. In all other situations, a new choice is necessary until a solution is found or the proof of its non-existence has been made.

5.2.2. *Sudoku and constraint programming*

In this section, we will see how to use a constraint solver to solve any sudoku grid. We will use the freeware constraint solver `choco` (available on `choco-solver.net`). This is a Java library. The remainder of this chapter will assume that the reader is familiar with the Java programming language. The code that will be given here is generic: it can be used to solve $n \times n$ grids.

Solving a sudoku with `choco` is quite easy. The first thing is to define a `Problem` (it is a `choco` class used to model constraint satisfaction problems):

```
int n = 9; // the size of the grid
Problem pb = new Problem(); // a new instance
```

The next step consists of defining the variables of the problem: one for each cell of the grid. We use for that the `IntVar` class that represents in `choco` integer domain variables. We will need to manipulate rows, columns and blocks. So, first we define the grid as a set of rows (columns will use the same references):

```
// rows and columns
IntVar[][] rows = new IntVar[n][]; // the n rows
IntVar[][] cols = new IntVar[n][]; // the new columns
```

Then we define n variables for each row (these correspond to each of the n columns). Their domain is the set of integers from 1 to n.

```
for (int i = 0; i < n; i++) {
    rows[i] = new IntVar[n];
    for (int j = 0; j < n; j++) {
    // creating an IntVar with name, min and max
        rows[i][j] = pb.makeEnumIntVar("("+i+","+j+")", 1, n);
    }
}
```

We need to define the columns (from the rows):

```
for (int i = 0; i < n; i++) {
    cols[i] = new IntVar[n];
    for (int j = 0; j < n; j++) {
        cols[i][j] = rows[j][i];
    }
}
```

Now it is time for the constraints. `choco`, like many constraint solvers, is equipped with a very powerful constraint, the `alldifferent` constraint, that ensures that the set of variables upon which it is defined each have a different value. This constraint implements the general principles developed in section 3.3.2 in Chapter 3 (page 44). One of these constraints is used for each row and for each column in the grid:

```
for (int i = 0; i < n; i++) {
    pb.post(pb.allDifferent(cols[i]));
    pb.post(pb.allDifferent(rows[i]));
}
```

Lastly, blocks need to be handled[7]:

```
IntVar[][] blocks = new IntVar[n][];
for (int i = 0; i < n; i++) {
    blocks[i] = new IntVar[n];
}
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        for (int k = 0; k < 3; k++) {
            blocks[j + k*3][i] = rows[0 + k*3][i + j*3];
            blocks[j + k*3][i + 3] = rows[1 + k*3][i + j*3];
            blocks[j + k*3][i + 6] = rows[2 + k*3][i + j*3];
        }
    }
}
```

7. This part of the code is not generic and is specialized for $n = 9$. We leave as an exercise to the reader the possibility of writing a generic code.

Once those blocks are defined, an `alldifferent` constraint will be posed:

```
for (int i = 0; i < n; i++) {
    pb.post(pb.allDifferent(blocks[i]));
}
```

Now, the complete definition of the sudoku grid is obtained. We now need to specialize the code for a specific grid. For each given, it is only necessary to enforce the variable associated to the cell to take the given value. Another constraint from `choco` is to be used: the `eq` constraint that imposes an equality relation.

Suppose for example that cell $(4, 6)$ contains a 5. The following code[8] will do it:

```
pb.post( pb.eq(rows[4-1][6-1], 5));
```

The final step is to solve the grid:

```
pb.solve();
```

That's it! We now have a Java program (using the constraint library `choco`) that can solve[9] any sudoku grid. Moreover, using the `alldifferent` constraint makes it possible to solve most of the grids with no enumeration at all. Only a few expert grids need this enumeration step (it remains very limited). Solving 9×9 grids remains instantaneous.

INFORMATION.– *This program can be seen working on the online applet:*

njussien.e-constraints.net/sudoku/eng-jouer.html

Finally, if one wants to print the solution, the following few lines can be used:

```
System.out.println("----- Solution -----");
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        // getVal gives the current value of a variable
        System.out.print(rows[i][j].getVal() + " ");
    }
    System.out.println();
}
```

8. Notice the necessity of shifting because of the numbering of rows and columns starting from 0.

9. It is instantaneous for 9×9 grids.

This page intentionally left blank

Chapter 6

Evaluating and Generating Grids

Software for sudoku is not only being able to solve grids. One should be able to both evaluate the difficulty of a grid and to generate valid grids. This is the topic of this chapter.

6.1. Evaluating the difficulty of a sudoku grid

Conventional wisdom states that the difficulty[1] of a grid is strongly related to the number of givens. This is not the case. There exist many easy to solve grids with only 17[2] givens (like the grid in Figure 6.1) whereas really hard to solve grids can exist with more than 30 givens (like the grid in Figure 6.2). Moreover, the position of the givens on the grid is also not a good indication of the difficulty.

EXERCISE 33. – Solve the grids in Figures 6.1 and 6.2.

6.1.1. *A subjective issue*

Actually, it seems perfectly normal that purely numerical criteria could not determine the difficulty of a grid. Indeed, rating sudoku grids is quite subjective. It responds to the player's feeling in front of the grid. A *difficult* grid will require subtle or complex techniques in order to be solved by players.

1. Difficulty – a subjective concept – is different from complexity – a mathematical concept.

2. This is the current known minimum number of givens for a valid grid. It has been mentioned in section 1.3.3 on page 20.

							1	
4								
	2							
				5		4		7
		8				3		
		1		9				
3			4			2		
	5		1					
			8		6			

Figure 6.1. *An easy grid with only 17 givens*

		3	7		9	5		
	2	7						
9				8			7	4
	9	8		3			6	
6				7	5	8		
		4	2	6		9		7
			5		7			9
			8					3
	5	9						

Figure 6.2. *An "expert" grid with 31 givens*

level	name	reference(s)	page
1	single position	rule 2.1	26
2	single candidate	rule 2.2	28
3	candidate lines	rules 2.3 and 2.4	30
4	multiple lines	rule 2.5	33
5	naked pairs	rule 3.1	37
6	naked triples	rule 3.2	38
7	hidden pairs	rule 3.1	37
8	hidden tuples (for $k = 3$)	rule 3.5	42
9	*XWing*	rule 4.1	50
10	naked tuples (for $k > 3$)	rule 3.3	39
11	hidden tuples (for $k > 3$)	rule 3.5	42
12	*Swordfish*	rule 4.2	51
13	constructive disjunction	section 4.3.1	54
14	*reductio ad absurdum*	section 4.3.2	55

Table 6.1. *Hierarchy of rules. Levels represents how difficult to understand and to use the rules are*

6.1.2. *A pragmatic solution*

Most sudoku producers use essentially the same technique: take a set of rules and techniques whose applying and understanding time increase; and rate a grid taking into account the global difficulty of the subset of rules necessary to solve it.

EXAMPLE.– Rule 4.2 which implements the *Swordfish* technique is much more complex to understand and use than rule 2.1 (single position). A sudoku grid that needs rule 4.2 to be solved is considered as more *difficult* than one that only needs rule 2.1.

The set of rules and techniques that was presented in the first part of this book can therefore be used to define difficulty levels. In this book, we use the *hierarchy* of rules reported in Table 6.1.

The difficulty level of a grid is defined by the minimal required level of rules that is maximally needed to solve the grid.

EXAMPLE.– Consider a grid G. Suppose that G can be solved in several ways (this is quite often the case):

– applying rule 2.1 (single position) several times and identifying an *XWing* configuration (rule 4.1) before getting back to rule 2.1;

– using only rules 2.1 and 2.2.

Therefore, in the first case, the maximal level is 9 (the level of rule *XWing* in Table 6.1) whereas in the second case, the maximal level is only 2. In order

4					2		7	6
9			8	6				4
	8		3		7			
2	6	8				7		9
5	4							
6				2	3			
		7		8	9	5		
			4	5				

Figure 6.3. *Grid to be evaluated*

name	minimal level
very easy	1
easy	2
medium	3 or 4
difficult	5, 6, 7 or 8
very difficult	9 or 12
expert	13 or 14

Table 6.2. *Difficulty ratings for the grid of this book*

to evaluate the difficulty level of grid G, the minimal value of these maximum level is considered. Here, the level is 2.

EXERCISE 34. – What is the level that is to be considered for solving the grid in Figure 6.3?

Once the minimal level is known, several difficulty ratings can be defined. In this book, we use the ratings presented in Table 6.2.

NOTE.– *In Table 6.2, rules 3.3 and 3.5 were not used to define the difficulty ratin when $k > 3$. Indeed, it is quite rare to be able to identify a quad (it is even harder to find a hidden one) in a sudoku grid. We chose to emphasize more visual techniques such as XWing (rule 4.1) or Swordfish (rule 4.2).*

INFORMATION.– *Implementing a sudoku grid evaluator is quite easy. The only point is to implement the rules in our formalism in order to be able to evaluate any sudoku grid. Notice that techniques such as disjunctive construction or reductio ad absurdum do not need to be actually implemented as, if no other rule is applicable, the considered grid is necessarily an expert grid.*

6.1.3. *The need of solving to evaluate*

In the previous section, we showed that, generally, evaluating a grid makes it necessary to solve it. Evaluating without solving is a research topic for sudoku.

6.2. Generating sudoku grids

The last point about sudoku is to generate valid grids. This is quite important for software manufacturers and for sudoku publishers.

INFORMATION.– *The Japanese editor Nikoli publishes only hand-made sudoku grids. Most of the editors usually use computer software or automatically generated grids.*

There exist two main approaches to generate sudoku grids (here solving a sudoku grid is not a problem anymore – using constraint programming for example, as shown in section 5.2.2):

– a top-down approach that starts from a solution grid which is *unbuilt* while the grid remains valid;

– a bottom-up approach that starts from an empty grid and adds givens one at a time until a valid grid is obtained.

6.2.1. *Top-down generation*

Top-down generation consists of performing the following steps:

1) generate a random solution grid;

2) (randomly) define a list of cells to erase;

3) choose a cell in this list (and remove it from the list);

4) erase the value of the cell in the grid;

5) if the resulting grid is invalid, put back the erased value;

6) if the list is not empty, go back to step 3; otherwise, the resulting grid is valid.

This technique supposes that certain operations are known:

– to generate a random solution grid: with constraint programming, a random solution search is needed;

– to be able to determine if a grid is valid or not (one and only one solution). In section 5.2.2, it has been shown that the `pb.solve()` method provides a solution to a constraint satisfaction problem. Knowing whether another one exists requires the use of the `pb.nextSolution()` which returns *true* when another solution exists. Such an answer would show that the grid is not valid.

With `choco` (and any other constraint solver) it is quite easy to define a valid sudoku grid this way.

6.2.2. *Bottom-up generation*

Bottom-up generation processes the grid the other way round:

1) start from an empty grid;

2) (randomly) choose a cell and give it a value compatible with the current grid;

3) if the resulting grid has more than one solution, go back to step 2; otherwise, the resulting sudoku grid is valid.

One question remains: how can a *value compatible with the current grid* be determined? It is merely necessary to make sure that the resulting grid possesses at least one solution. With a constraint solver, calling the `pb.solve()` method should suffice!

This process is quite simple to express but is less easy to implement. Indeed, it makes it necessary to solve the same grid several times (in order to check compatible values) and to go back to a previous state when no compatible value is found. The constraint solver needs to provide mechanisms to explicitly manipulate backtracking. This can be tricky. No more details will be given here[3].

NOTE.– *No process is the best. Each one has its pros and cons. The top-down approach is probably the easiest to implement but does not seem natural to the constraint programming practitioner.*

6.2.3. *An open problem*

Directly generating a grid for a specific rating is unknown at the time of writing. This is still an open question.

INFORMATION.– *When generating thousands of sudoku grids, an interesting phenomenon occurs. Most of the generated grids are easy or very easy. Expert grids show up quite often. But,*

3. The reader can refer to the `choco` website: choco-solver.net

the most difficult to generate grids are difficult and very difficult grids. It would be interesting to work on the way grids are scattered through the space of possible grids.

Techniques and rules of this book where used to generate the 125 grids that follow. More can be found on:

http://njussien.e-constraints.net/sudoku/eng-index.html

This page intentionally left blank

PART 3

Training

This page intentionally left blank

Chapter 7

10 Very Easy Sudoku Grids

数独 1

7			2	6		3		
			1	5	8			
8			3	4				1
2			4	7		1		8
1								2
	7	8		1		4		
		3					6	
				3	1	7	4	
5	6			8		2		

数独 2

	5	2			3			4
	4	1		6			2	
					8	1		
4		7		2	6		5	8
	6		7				1	
		5	1			2	7	
	7	4	6			3		
					5			
6		3			1			

数独 3

7		6						
8	2			1				
	3	1		6	2	8		5
		4		8		5		
3	5					6		2
	7				3	4	1	
6	1	2	5			3	7	4
			1		7			
	4							

数独 4

7					4			8
8	4			1		3		
2				3			6	
							3	2
1		4		5	2			7
5		3						1
	5			2			1	
6							4	
	7	8			3		2	

数独 5

		8	1					
	2						4	7
6								
	7				2	1		
8	6				1	7	3	
		5	7					
		2	4					
			2	8				
3		1			5		6	

数独 6

	4			8	6			
7	6							
	1	3		5		4		
	8	5			3	2		
1					8			4
	2	7	4				1	
			2		7	5	3	1
						6	8	2

数独 7

	8	6			2	5		
7							1	
	2	4						
		5		8	7			
		3	5		1		2	6
		8	6	3	4	7		5
					3			
3		2			6		7	
			8	1				

数独 8

	1	7		4		3		6
						2		
2	6							8
	4			6	3	7	1	
3			2			6		
6		8		5	1			
							2	1
		3			4			
4	8	6	1	2	7			

数独 9

	8	4	2	6	1	5	7	
	2		3		7	8		
3		1					2	
7	5	6		4		3	1	
	1		8		3			5
2			1					
	6		4					
	4					1		
		3	6		8	4	5	

数独10

3				5			2	7
	1	8			2		3	
					6		1	5
	6		8			4		
			7					6
4		3	5		1			2
			4			2		
		5						
1		2			8		4	3

Chapter 8

10 Easy Sudoku Grids

数独 11

8	4		1	2				
3		1	7	5		8		
			6					
		2				6	4	8
						5		2
				6		3		
4		7	8		3			6
	5	3				1		
				7	1			

数独 12

			1		7	6		
1	7		8	6			3	
						2	1	
			7		1			4
	1	6						
				8		1	5	
3					2		7	
4					5			
7				1				5

数独 13

				3		5	4	
			8	5		2		
	4		7			1	3	
1			2	4		3		
		2						
	8			1		6		
				8				
5		8	1	6				7
	1		5	7				

数独 14

7		4						
6		1	7		2		5	
				3				
	5	3	4	6		2		8
				7		6		4
		7	8			1		
	4				1			
2		8			4			
				5				2

数独 15

6		1			8		5	
7				3				8
3				1				
1				2	3	8		
	5			7				
	7			4		6	1	
			7				2	1
						5		4
		2	4	8				

数独 16

3		7						8
	1						6	
	2	4	6	5				7
		2				1	8	
			5	2			7	4
				4		3		2
2	8			1			3	
5							2	
		6			2	5		

数独 17

5		2						
			8		6	9	5	
					3		7	8
	2							
							8	9
	7		9			5		4
			4		9	3	6	5
4		3		6				
			7	3	8		9	

数独 18

	8		6			2		3
5		4	2		9			7
					8	9	4	
			7	5	2		6	9
		5	4	3				
7			9	8	6			
2		9	5		7	8		
4			8					

数独 19

	2	9	8	3				5
7		8		2		3	9	6
6					5	4		
		6		9	2			
8		7	3		4	2		
			7	5	8		6	
						9		
								7
	7	3		8				2

数独20

7	3		2		8	9		5
	9	8	6	5		4	2	
8			7		4		9	
		7		6				2
5		9	4					
6					5	3	4	
					9			8

Chapter 9

25 Medium Sudoku Grids

数独 21

			5				8	
5	8	4					7	
6		7					9	3
						3	6	7
	3			2	9			5
7			3				2	9
			9			6	4	
				3	7			
	2			8				

数独 22

	6	4					2	
2			9					3
		7		6			9	
		3		8		2		9
5					3		7	
			7		9			
6				3	7	8	5	
	7							2
4	2						3	

数独 23

					6		4	
6	4	9	7					2
	5		9	3		8		
2		8					3	
		6				9		
						2	5	
			3	9				
	3	7	4					
9				7	8	4		

数独 24

		8	4	9	5		6	3
	4				3	7		
	3				7		9	5
	6							
		9	6			8		
8			3					
	2			8				9
5		7						4
				4	6	2	5	

数独 25

					9			8
				3		4	7	
4				5	8			9
		5		8			4	2
	4				7	6		5
					2	9	3	7
	5	8	9			7		
7		4		2				
6	9	2						

数独 26

	4		6	7			5	
8			3					
		6		9				
			7	8	2			9
							3	
	9	5		4	3	7	6	8
3		4	2	6	9			
5					4			7
	2					4		

数独 27

		4					2	
6				2				
						7	3	
8		6			9		4	
						8		
9	5		8		7			
5	8	2		6	3		9	
							7	
		9		5				8

数独 28

	5	6				7		9
					5			
8	3			9	7		4	
			9			5		8
9	2	7	3					
					4			
		5	7	2	6			3
	9							7
2			8	3				6

数独 29

				7		5		
4	8	3	9	5			7	6
				6	2			3
				8		9		
	5	9					4	
8							3	5
2			7	4			9	
					8			4
7	4		3	9				

数独 30

		5			4	3		7
6			2			4	8	9
8				7				
	9				2	8	7	
			9	3			6	
5			7	6				
	8		4	9				6
4	6	7					3	

数独 31

	8		5	7		4		
3						6	2	
					8		9	
	4					8	7	2
						9	6	
	9		6	5	7		3	
9	6	8			4	2	5	
	7	5						
4						7	8	

数独 32

		8	5		9	7		2
			8	7			9	
			6	4				
	9					3		
6		7			4	8	2	9
	4	5						7
	5	4				9	8	
				3		6	5	

数独 33

		2	7	4			6	
		8		9	6	5		
		3	2					
		7					3	5
6				2				7
								9
2							9	
			5	8	9	4		
8	7							

数独 34

						5		3
9		2		7		8	6	
	8	3	2				9	7
		7				3		
			7					
6							4	
7				2	3			4
	5	9	8	6				
					9			

数独 35

		4				9		
		2		5			8	
3		8	7		9		2	6
	2	5					6	4
6								
	7					5	9	8
	8	7	6					2
9				7	3			5
				8				9

数独 36

9	6		8		2			3
	2	7		6		8	4	
8				5				9
6	5	8		2		3		
	9							4
	3	5	2		9			
4							5	
				7	5			

数独 37

			8		2		6	5
5				9				
7					6	9		
9		3	5	8		7		
			4				8	
4		8	7				3	6
							7	
			9	4				
		2		3				

数独 38

	8				5			
	7					4	8	5
3	9					7		6
7	6	3						
5			9	2			3	
9	2				3			7
	4	6			9			
			7	3				
				8			2	

数独 39

	2					4	6	
3					9			2
	9			6		7		
9		2		8		3		
	7		3					5
			9		7			
	5	8	7	3				6
2							7	
	3						2	4

数独 40

	4		6					
2					7	9	4	6
		8		3	9		5	
	3					8		2
		9				6		
	5	2						
				9	3			
					4	7	3	
		4	8	7				9

数独 41

3	6		5	9	4	8		
		7	3				4	
5	9		7				3	
							6	
		8			6	9		
					3			8
9				8			2	
4						7		5
	5	2	6	4				

数独 42

8			9					
	7	4		3				
9			8	5				4
2	4			8		5		
5		6	7				4	
7	3	9	2					
		7			9	8	5	
				2		4		7
						2	9	6

数独 43

	5			7	6		4	
					3			8
				9		6		
9			2	8	7			
	3							
8	6	7	3	4		5	9	
			9	6	2	4		3
7			4					5
		4					2	

数独 44

	2					4		
				2				6
	3	7						
	4		9			6		8
		8						
			7		8		5	9
	9		3	6		2	8	5
	7							
8				5		9		

数独 45

9		7				6	5	
			5					
	4		7	9			3	8
8		5			9			
					3	7	2	9
			4					
3			6	2	7	5		
7							9	
6				3	8			2

This page intentionally left blank

Chapter 10

25 Difficult Sudoku Grids

数独 46

							4	
			3	6	4	9	8	
5						6		
			6			7	3	4
3			5			2		8
8	2		7		3	5		6
4			9	5	6		7	
		9						2
	5	3	4					9

数独 47

				7	4			
			6		9		4	5
4						9		
	5				3			
9		7	2			6		
6			9					4
	8	2	7					
	4	6		5		8	9	7
		9				2	3	

数独 48

6							3	4
	8		9			6	7	
		7				8		9
	7				9		8	
	9				4	3		
5		8	6	3	2			7
		4		6	7			8
	6		8				5	

数独 49

2	8							9
5	4	6		9		2	8	
9		7				4	5	
			5	7				
6			2	4			7	
8			3				9	
				3		9		
			7	8				
	3		9		4		2	

数独 50

4	8						7	
	9			2				
7	6				5			
				9	7	3		2
9	5		3	4		7		
			6	8			9	5
				7			2	8
2						9		
6	4	8						

数独 51

				2	8		4	
6	2		4			3	9	
					7	5		
9		4			2	8	6	
				5	4			
5								7
		6						
				8				
7		8			9	6	5	2

数独 52

8			3			6	4	5
				9				
7						2		9
				4	5			
		3	6	7			2	
		2		3			8	
	9		4				6	
4	5					9		
	8			6		3		

数独 53

3	5				9		8	
		4			5		6	
7				8		5		
	7						5	
		5	6		7		9	2
8	9					6		3
4	6					3		
9	2	8						
								6

数独 54

4		9	5			2	6	
								3
		6	7		8	3		
							4	
2			3	6		7	9	
		2	9	5	3			
	5		4					
9			6		2		8	7

数独 55

						7		
			7	8	9			
				6	5	3		4
							4	5
			8				7	
	5			4	2	8		
4				9				
		8	6	2	4		5	
9	3			7	8		2	6

数独 56

				9	5			7
		5	8			3		
4		7		2		5		9
2		6		7				
3	7		9		8			
5	9					7		
				6	3	9		
	3					2		
8					9			

数独 57

5	9			6				3
			8		5			6
8	6		9	3	2	5	4	7
				2		4		5
			3	9			7	
						9		8
3			2				6	
	7		4	8	9		3	

数独 58

			6		9			
			2			4	5	
			5				8	7
	9				2			
4								5
						6		9
7	4		8				9	2
6		9			5		7	8
2		8	7	9		3	6	

数独 59

			2		7			
	5						2	8
	2	3				6		7
				4			6	
			7	2	8	9		
	3			9				2
8				5	9			
		9			4		8	5
		4					7	

数独 60

							3	
9								2
5	7		3	8			6	
7		2		3				5
8					4			
			6	2		7	9	
	4			6	9		7	
					7	9		8

数独 61

					3			9
9					7			
4		8	9					5
	8	5			4			3
			3			2	9	8
		9	6			5		
	2		7		9			
	9	7	4				8	2
			8	5			7	6

数独 62

	9							6
5					8			
	4	8					2	
8				5		3		
9		6						
		2	8	7		6		9
					5	7		
	2			9		4		3
7	6	3						5

数独 63

		4	8	3		5		
5			2					
3	9							
4		9	7	5	6		3	
		5					6	
		6	3				4	
			5	2	7	6	9	
		7			3		8	4
9	2		6	8	4			

数独 64

			6	9		4		
	4	8			5			
9	6	2		7				
		7	9	2			6	
	5							7
					3		9	4
2	8			6		9		
3	9						4	
	7		4				5	

数独 65

			5					6
6				9	7		8	
		4	8			7		
8			6				9	
		6	3					
		7	2	4	9			
				3		8	6	
5					8		7	3
		8	7			9		4

数独 66

			8	6		9	5	2
3							4	8
						7	6	
9	7		3	2	5			
	8	3		4	7		9	
4								
		9				4	2	
2			9	7		5	8	
								9

数独 67

6	2			9		7		4
4				5		6	9	8
8								
			6	3				
		7	8	4	9		2	
					7	5		
	9			7	3			
		2	9					7
		8	5		2			

数独 68

		5	3					
	1	7			6		8	4
	6		2	8				
8	5	4	1			2		3
				2				
						4		
3								5
			6	5				
	4	2	8			6		1

数独 69

1		8						3
				1				
5	6	4			7			2
		7		4			2	
		1					5	6
	4				6		1	
	2			7		8		
	8			3	4	7		
			5	6				

数独 70

		5		2				3
	4		5			6		
	2		1				5	7
4								
						2	8	1
		7					4	6
7		4					1	2
8	1		3		4	5		
	5						3	

This page intentionally left blank

Chapter 11

25 Very Difficult Sudoku Grids

数独 71

6								
3				2		1		
							2	7
1						4		
	6		7	3			1	5
		5		4			6	
					3	5		
		4	2			8	7	
2					1		4	

数独 72

			3					1
5		6		7	2		4	
	2	1			6			
		3		1			5	8
8	7	5		2				
		4					2	
		2						6
1					3		8	
	8	7			4		1	5

数独 73

	5	6			1	2	4	7
	3			8		5		1
	4		2				3	
	1				6	4	5	
4	2		7	1			8	
						1		
1					3	7		
6				2		3		8

数独 74

	2		1		3	8		
4	8			7		1		2
				2				7
			3		1	5	2	
	1		6					4
		4	7					
			2					
				8	6			
7	5						3	

数独 75

		2						9
3						4		7
		8	9	3	4			
	7					9		5
			7				2	
	5		6		9			
	3		4		5			6
		7			6	8		
9			2	8	3	7		

数独 76

9		8		7			4	2
			9				6	
		6	8		4		9	
5	8	3						4
				6	3			7
3	5				2	8		
	6			3	7			9
2		7			5			6

数独 77

		8	7					6
4					9			
			5	4	6	9		
					3		5	
		3			7	6		
							8	9
	7		4		2			5
8			9		5		2	3
2		9	3		8	7	6	

数独 78

							5	9
					6	3		
			7					8
		3	5		7			
			6	9				7
5				8				
9		6				4		
		7	3			2		
3		8		2	5		9	

数独 79

2			3			8		
	8							2
4						9	3	
					3	6		
	9		5	8			7	
				9		4	8	
		8	2		4			6
9	6				8		4	
5			9	3		7	2	

数独 80

	9			7		6		
	8	5			3			
3			5		6		9	
	3							
8			9	5		7		3
			2				8	
				6	9			
	7	9	4					
4	6		7			2		

数独 81

		6			8			3
	2	4			9			
			4	6				
	6	8	2		3		7	4
						6		
3	7	2			4			
		5	9		7			6
		3				5	8	
		7				4	9	

数独 82

	4		5	9				
6		7				9	2	
9			6		4			5
		6						
3		9			8	5		
	7	5	4	2				3
				5				
			3	8	7	2		6
			9	4		7		

数独 83

4				2				
								9
5		2	9	4		8		
	2		8	6			7	
8			5		7		3	
9	6	7				5		
	8				6	2		
		4					9	
		3		5		7		6

数独 84

						5		
2	8							7
7		9			2			
				4		3		
		4			9		6	2
				8			9	
3		6	9			8	5	
4			5			7		
	9	7	3					

数独 85

							9	
7					9			
	5		3	4				2
	7	4		8				
8	2		9	7				6
		9			6			
		5	7					8
3				5		2	4	
2	6				8			5

数独 86

3							5	
8			6	3		4	9	
	9			2	5			6
			9		7			
	8					6	7	
				8			3	
				6		5		
							4	
	7	4	8					9

数独 87

		9	5		4			2
	6		9	2		8		5
4	8		6	7				
	3						8	9
		7						
9	4		3		8	6	5	7
8			2				3	
	5			9			2	

数独 88

	9	3	2					
7		6					2	5
	5	8	7	3		9		6
		9		7				4
	2		8				9	
8							3	
		4						
6	7	5					8	9
			4	9			6	

数独 89

3	8						6	
			7					8
	9	2						5
		7		4				9
8	3			2				
		9			8	3	4	7
						6		
	2	8	5				9	
	6		3					

数独 90

	3	1			4			5
							8	
	6		3	2				1
6			7	4	5	1		
7			2					
1			8		6	4	3	
2		8		3				
					7	5		3
	7		1					

数独 91

		8	4	1		3	6	
	4	6						1
					2			
			5	3	8	7		6
		3		7		4		
	1							2
		2	3				7	4
				4	5		2	
		1	8		7		5	

数独 92

		4		7	5	1		
			4	8		2		5
1			3				4	
4	1		2	5	8		3	7
6								
5		3	7	1	6			
		2	1				7	
					3			
	6		8	2				

数独 93

2		1					3	
		5	1					
	7			8	6			
	4		5					3
			8			2	1	
3				7			4	5
			2	3	4			7
			7		1	5		

数独 94

	8				4			
7			8	6		2	3	
1		2	3			6		4
				2	6			7
			7			1	2	
		7	1		8		5	
					2			
	2			4			1	
6		8	5	1				

数独 95

							7	
1						2		
		4	8				3	
4	7				1			
		3			4		5	
5			3	6		8	1	
	5			1				
	2	1	4	3				7
7			6				2	

This page intentionally left blank

Chapter 12

30 Expert Sudoku Grids

数独 96

	5			4		8		3
	1	4						
			8	7		1		
	4		1		2	7		6
		2	3				1	
								2
			6					7
		3	7				8	4
2		5						

数独 97

1								8
	7			3				
	4		2		1			
	2				3		6	
4			7		8		2	5
				1				
3	1	4	5	8				
	5	2						
8		6				1		

数独 98

			4		5			8
			2	6	3		4	1
	6			1	8			
				3				
1	3			5	2			6
4	2	7	6					
2		1						3
	4	5						
8					1		6	

数独 99

		1	7		5	6		2
		4						
	8		1		4			
	4							8
1	3			2			5	
2			6			7	3	
					1			
	7		2					1
				3			8	

数独 100

		2		5			8	1
3	5				7	4		2
	8			3				
8		7		1	3			
			2					
						6	5	
7	4				8	5		3
			3			2	4	
2					1	8		

数独 101

		5						4
	2				4			
8		4	6	2	7	1		5
				4	8			
	3	8			5		6	
	7		8	5		3	1	
		3				5		6
		6	1	7				

数独 102

2	4					7		
1	5						3	
					2		4	
	3			6	1		2	
	6		2				5	
	8				5			7
	7					5		6
3	1			5	6		7	
	2	5	4			3	1	

数独 103

5		6		7	1	3		
			4		3		6	
								8
			2				3	4
2								1
3				1	5	2		
8	5	3		2	7			
1	4	2			6			
					4			

数独 104

1	8	6			3		4	
				8	7			
	2	5		6			8	
		4		1		3	2	
				4		5		
3			6				1	8
5				2		7		
	3							
		2	1					4

数独 105

4	5					3	2	
7								9
9	8						4	5
8					4			
			3	5	8			7
			6		7			
3	7		5					
	4	8	9			5	7	
	2	9		4	6		8	

数独 106

3					8	2		5
6		2		7				4
7	4	8	2		5	9		
	9		7			6		
	3	4	8	9	6			7
		7					4	
					3	4		
		5					6	2

数独 107

	9		3	8				
		5						
			7		9	5		6
	7					9		
3	2		4					8
5	8				6			
						6	5	
				9			8	7
7	6					3		4

数独 108

6	9			8			3	
		5	2		3	4		
		2	9					
						3		6
				4	2			5
		9						8
				7		5		
7					9			
		4	8			9		7

数独 109

				6			7	
			8			9		
2	8		7		9			
	6	8					2	
5		7		9	4	6	8	
		4						
9		6	4	7			5	
	4					7	9	
	3						6	

数独 110

		4	2		8	9		5
9		8				6		
6			9		5	8	4	7
			7					8
2	3		8					
					6	4		
8		7	6	5				
	9				2			
	6		3					9

数独 111

						8		5
3				2	6			7
4					3			
					8			
7				9				8
9		8	3		4		6	
2	3		9					
							9	6
	6		2		7		5	

数独 112

			9			2		4
							5	8
5	2					7	9	6
	9							
3		2		8	5	6		
		7		4			8	
8		9					6	
	7						3	
					2	9		

数独 113

		9		4		2		
							6	5
					7	8		9
4						6	8	3
	5	8			4	9	7	
	7							
	9	2					4	
8	4	7		6	9			
6		5			2			

数独 114

	7			2				
8					9		3	
		9						
4			3	7		8		
	8			5		9	7	
					2		6	
9		6					2	
								6
3		5	4		8			9

数独 115

		9	2			8		
7			8	6				
		2	5		7	3	6	
			4	5				
8								
	9	7				2		3
	7						2	
		3	6		8	7	5	
	5			2	9			8

数独 116

9	3							4
			5		4			7
2	5		7	9			3	
		5		4			7	2
	6	2						
4	8	3	9		5	2		6
		6					8	
					6			5

数独 117

6			7	9			8	5
	5	4		3		7	9	
			5		4		3	
		5			3		2	
8				5			4	
	4	9		8			7	
		8		6				
				7		9	5	
			3			8	6	

数独 118

		6						
		4				9	6	8
	8	3				2	5	7
	9	5	8			7		
					9	8		
8				7	6			
					2			
6		7		4				
	3	9	7			5		4

数独 119

9					6			8
							7	
	8		3			5		
4				9	2	7		
	6		5					
	9		7	8				6
	4			2			8	5
	2	3						
		7		6		9	2	4

数独 120

				3		5	1	
2	3					6		8
	5							7
4			3	2	6			
6				5				
	1	5	4	7				
1	2							
8	6	3				2		5
		7			2	1	3	6

数独 121

8				3			6	5
4			6					
	6				4	1		8
	7			4		5		2
				1			3	
				2	3	8		
5			3	6		2	8	
				7	1	6		
						3	4	7

数独 122

6	3			2		4		
	2	5						
7		4			1	5		
			4			1		
	1							2
				6		3		7
							5	
4			2	8	3			1
3			5	7				

数独 123

3			2	5	4			
								7
1		5			7		6	
	1			8			7	
		3		6				2
2						1	8	
6			1			8	5	
								1
	3							4

数独 124

4	1	5		2	8			3
	3			4			1	
				6		1		
		3		1				4
		6				3	2	
		4	6	3	2			
7	6				4	2		
		1		5		8		

数独125

1					2	3		5
						6		
			6			2	4	1
	8		4			5		2
		5						
7		4		2	3	1		8
		3					2	6
4	1			7				
		2		8		4	1	

Appendices

This page intentionally left blank

Appendix A

Corrections

EXERCISE 1 PAGE 27

In cell $(8, 5)$.

EXERCISE 2 PAGE 27

Put a 6 in cell $(2, 3)$.

EXERCISE 3 PAGE 28

5	9	7	2	3	6	4	1	8
8	1	6	9	7	4	3	5	2
3	4	2	5	1	8	9	7	6
6	2	4	1	9	7	8	3	5
1	5	3	4	8	2	6	9	7
7	8	9	6	5	3	2	4	1
2	6	1	7	4	9	5	8	3
4	7	8	3	2	5	1	6	9
9	3	5	8	6	1	7	2	4

EXERCISE 4 PAGE 28

The single candidate is 5.

EXERCISE 5 PAGE 28

Cell $(3, 3)$ for a 9; cell $(9, 4)$ for a 9; and cell $(6, 5)$ for an 8.

EXERCISE 6 PAGE 28

The grid is completely solved! The result is:

6	4	5	7	9	3	2	8	1
2	1	7	8	4	6	5	9	3
3	8	9	5	1	2	4	6	7
7	5	8	6	2	1	3	4	9
9	2	3	4	5	7	8	1	6
4	6	1	3	8	9	7	5	2
1	9	4	2	7	8	6	3	5
5	7	6	1	3	4	9	2	8
8	3	2	9	6	5	1	7	4

EXERCISE 7 PAGE 29

The grid cannot be solved only using the two first rules. An exhaustive application gives:

	6	3	5		1		4	7
	7	5	8	6				
	1	2		7			5	
5	4	8	2	1	9	7	6	3
1	2	6	3	4	7	5	9	8
3	9	7	6	8	5	1	2	4
2	3	9	7	5		4		
6	8	4	1				7	5
7	5	1						

EXERCISE 8 PAGE 29

In cells $(2, 6)$ and $(4, 1)$.

EXERCISE 9 PAGE 30

The result is:

6	7	9	8	1	2	4	3	5
3	8	1	4	5	9	6	2	7
5	2	4	3	6	7	1	8	9
9	1	7	5	2	8	3	6	4
4	3	2	7	9	6	5	1	8
8	6	5	1	4	3	9	7	2
7	5	8	6	3	4	2	9	1
2	4	3	9	8	1	7	5	6
1	9	6	2	7	5	8	4	3

EXERCISE 10 PAGE 31

9	6	3	5	2	1	8	4	7
4	7	5	8	6	3	2	1	9
8	1	2	9	7	4	3	5	6
5	4	8	2	1	9	7	6	3
1	2	6	3	4	7	5	9	8
3	9	7	6	8	5	1	2	4
2	3	9	7	5	6	4	8	1
6	8	4	1	3	2	9	7	5
7	5	1	4	9	8	6	3	2

EXERCISE 11 PAGE 32

For block B_7 and value 8.

EXERCISE 12 PAGE 32

4	5	9	1	7	2	3	6	8
3	1	8	5	9	6	4	7	2
6	7	2	4	8	3	1	5	9
5	3	4	7	6	8	9	2	1
2	9	6	3	1	4	7	8	5
7	8	1	2	5	9	6	4	3
9	4	5	8	3	7	2	1	6
1	6	7	9	2	5	8	3	4
8	2	3	6	4	1	5	9	7

EXERCISE 13 PAGE 32

Value 3 can be assigned to cell $(4, 1)$. The following regions can be checked in the following order: R_2 (for value 1), R_9 (for value 1), C_1 (for value 9), C_6 (for value 8), C_8 (for value 2), C_9 (for value 8) and finally R_5 (for value 8). The resulting grid is given in Figure A.1 (values in gray are impossible).

EXERCISE 14 PAGE 33

The rule can be written as:

– **Parameter**: two blocks B_{b_1} and B_{b_2} sharing a common line and a value v. Let b_3, j_1, j_2, j_3 be values that verify the relation: $B_{b_1} \cup B_{b_2} \cup B_{b_3} = C_{j_1} \cup C_{j_2} \cup C_{j_3}$.

– **Condition**: $\texttt{where}(\{v\}, B_{b_1}) \subseteq C_{j_1} \cup C_{j_2} \quad \wedge \quad \texttt{where}(\{v\}, B_{b_2}) \subseteq C_{j_1} \cup C_{j_2}$

– **Deduction**: $\forall (i, j) \in B_{b_3} \setminus C_{j_3}, (i, j) \neq v$

8 9	6	7	5	4	2	3 8 9	3 9	1
2	3	4	6 9	1 8 6	1 8 9	5 8 9	5 9	7
1 5 8 9	1 8 9	1 5	3 9	1 7 8 3	1 7 8 9	2	4	6
3 8	4	6	7	2 3 8	5	1	2 3 9	2 3 8 9
3 5 8	2 7 8	2 5	1	9	6	3 5 7 8	2 3 5	4
1 3 5 8	1 2 7 8	9	2 3	2 3 8	4	3 5 7 8	6	2 3 8
4	5	8	2 6 9	2 6 7	7 9	3 9	1	2 3 9
7	1 2 9	3	4	1 2	1 9	6	8	5
6	1 2 9	1 2	8	5	3	4	7	2 9

Figure A.1. *Resulting grid for exercise 13*

EXERCISE 15 PAGE 33

6	1	4	8	3	7	9	5	2
5	3	2	4	9	1	8	7	6
7	8	9	2	6	5	3	4	1
9	4	1	7	8	6	2	3	5
8	6	5	1	2	3	4	9	7
3	2	7	5	4	9	1	6	8
1	7	3	9	5	2	6	8	4
4	5	6	3	1	8	7	2	9
2	9	8	6	7	4	5	1	3

EXERCISE 16 PAGE 33

9	6	3	5	2	1	8	4	7
4	7	5	8	6	3	2	1	9
8	1	2	9	7	4	3	5	6
5	4	8	2	1	9	7	6	3
1	2	6	3	4	7	5	9	8
3	9	7	6	8	5	1	2	4
2	3	9	7	5	6	4	8	1
6	8	4	1	3	2	9	7	5
7	5	1	4	9	8	6	3	2

9	7	3	4 8	5	1	2	4 8	6
2	5	4	6 8	3 6	9	1 3 8	7	1 3
1	8	6	2 4	2 3 4	7	5	3 4	9
3 4	1 9	7 8	4 5 7 9	1 4	6	3 1 4 8	3 4 5 8	2
6	2	5	4 7	8	3	1 4	9	1 7
3 4	1 9	7 8	4 5 7 9	1 4	2	3 1 4 8	6	3 1 5 7
5	6	1	3	7	4	9	2	8
7	4	2	1	9	8	6	3 5	3 5
8	3	9	2 6	2 6	5	7	1	4

Figure A.2. *A difficult grid solved using medium rules*

EXERCISE 17 PAGE 34

9	7	3		5	1	2		6
2	5	4			9		7	
1	8	6			7	5		9
					6			2
6	2	5		8	3		9	
					2		6	
5	6	1	3	7	4	9	2	8
7	4	2	1	9	8	6		
8	3	9			5	7	1	4

The grid in Figure A.2 gives the remaining list of candidates.

EXERCISE 18 PAGE 38

9	7	3	4	5	1	2	8	6
2	5	4	8	6	9	1	7	3
1	8	6	2	3	7	5	4	9
3	1	7	9	4	6	8	5	2
6	2	5	7	8	3	4	9	1
4	9	8	5	1	2	3	6	7
5	6	1	3	7	4	9	2	8
7	4	2	1	9	8	6	3	5
8	3	9	6	2	5	7	1	4

6	4	5	7	1 2	9	1 2 3	8	1 2 3
8	1 3	1 2 3 7	4	1 2	5	6	1 7	9
1 2	9	1 2 7	3	6	8	1 2 4	5	1 2 4 7
3	7	9	1 5	8	2	1 4	6	1 4 5
4	8	1 2	1 5	7	6	9	3	1 2 5
1 2	5	6	9	1 3 4	1 3 4	8	1 7	1 2 4 7
7	2	8	6	1 3 4	1 3 4	5	9	1 3
9	1 3	4	8	5	1 3	7	2	6
5	6	1 3	2	9	7	1 3	4	8

Figure A.3. *Resulting very difficult grid*

EXERCISE 19 PAGE 39

In cells $(6, 5)$, $(6, 6)$, and $(6, 9)$ for values 3, 7, and 9.

EXERCISE 20 PAGE 39

In cells $(7, 9)$, $(8, 9)$, $(9, 7)$ and $(9, 9)$ for values 3, 5, 7, and 9.

EXERCISE 21 PAGE 42

The naked triple rule (rule 3.2) can be applied for values 4, 6, and 8.

EXERCISE 22 PAGE 43

9	7	3	4	5	1	2	8	6
2	5	4	8	6	9	1	7	3
1	8	6	2	3	7	5	4	9
3	1	7	9	4	6	8	5	2
6	2	5	7	8	3	4	9	1
4	9	8	5	1	2	3	6	7
5	6	1	3	7	4	9	2	8
7	4	2	1	9	8	6	3	5
8	3	9	6	2	5	7	1	4

EXERCISE 23 PAGE 43

6	4	5	7		9		8	
8			4		5	6		9
	9		3	6	8		5	
3	7	9		8	2		6	
4	8			7	6	9	3	
	5	6	9			8		
7	2	8	6			5	9	
9		4	8	5		7	2	6
5	6		2	9	7		4	8

The grid in Figure A.3 gives the remaining candidates for each cell.

EXERCISE 24 PAGE 44

Here is the resulting partitioning:

– columns C_5 and C_8 given;

– columns C_4, C_6 and C_9 sharing values 3, 6 and 7;

– columns C_1, C_2, C_3 and C_7 sharing values 2, 4, 5 and 9.

EXERCISE 25 PAGE 47

Two other auto-sufficient sets exist: $\{c_1, 1\}$ and $\{c_4, c_5, 4, 5\}$.

EXERCISE 26 PAGE 50

Value 5 is only possible on rows R_3 and R_7 for columns C_3 and C_8. Candidate 5 can be removed from all the other cells of those rows. Then, cell $(3, 2)$ is assigned with the value 7.

EXERCISE 27 PAGE 50

6	4	2	9	3	1	7	8	5
8	9	1	5	6	7	2	4	3
3	7	5	4	8	2	1	6	9
7	3	4	6	2	8	5	9	1
2	8	6	1	5	9	3	7	4
5	1	9	7	4	3	6	2	8
1	2	7	3	9	4	8	5	6
9	5	8	2	1	6	4	3	7
4	6	3	8	7	5	9	1	2

EXERCISE 28 PAGE 52

Candidate 9 is only possible in rows R_3, R_4, and R_9 for columns C_1, C_5, and C_6. Candidate 9 can then be removed from the candidate lists of all the other cells on those

2 6	2 6	1	5	3	4	9	7	8
7	5	8	9	2	6	4	3	1
4 3	4 3	9	7	1	8	6	2	5
5	9	6	4 2 8	4 8	7	3	1	4 2
1 4 8	1 4 8	2	3	6	5	7	4 8	9
4 8	7	3	4 2 8	9	1	2 8	5	6
1 2 3 8	1 2 3 8	4	1 8	7	9	5	6	2 3
9	1 2 8	5	6	4 8	3	1 2 8	4 8	7
1 3 6 8	1 3 6 8	7	1 4 8	5	2	1 8	9	4 3

Figure A.4. *An expert grid being solved*

rows. Then, rule 2.1 applies for column C_7 and value 9 can then be assigned to cell $(6, 7)$.

EXERCISE 29 PAGE 52

3	7	9	6	4	8	2	5	1
1	5	6	7	3	2	4	9	8
8	2	4	1	5	9	7	6	3
7	6	2	4	9	1	3	8	5
4	9	8	5	2	3	1	7	6
5	1	3	8	6	7	9	2	4
2	4	7	3	8	5	6	1	9
6	8	1	9	7	4	5	3	2
9	3	5	2	1	6	8	4	7

2	6	1	5	3	4	9	7	8
7	5	8	9	2	6	4	3	1
4 3	4 3	9	7	1	8	6	2	5
5	9	6	4 2 8	4 8	7	3	1	4 2
1 4 8	1 4 8	2	3	6	5	7	4 8	9
4 8	7	3	4 2 8	9	1	2 8	5	6
1 3 8	1 2 3 8	4	1 8	7	9	5	6	2 3
9	1 2 8	5	6	4 8	3	1 2 8	4 8	7
6	1 3 8	7	1 4 8	5	2	1 8	9	4 3

Figure A.5. *An expert grid being solved*

EXERCISE 30 PAGE 52

		1	5	3	4	9	7	8
7	5	8	9	2	6	4	3	1
		9	7	1	8	6	2	5
5	9	6			7	3	1	
		2	3	6	5	7		9
	7	3		9	1		5	6
		4		7	9	5	6	
9		5	6		3			7
		7		5	2		9	

Figure A.4 gives the candidate lists for each cell.

EXERCISE 31 PAGE 55

2	6	1	5	3	4	9	7	8
7	5	8	9	2	6	4	3	1
		9	7	1	8	6	2	5
5	9	6			7	3	1	
		2	3	6	5	7		9
	7	3		9	1		5	6
		4		7	9	5	6	
9		5	6		3			7
6		7		5	2		9	

Figure A.5 gives the candidate lists for each cell.

EXERCISE 32 PAGE 55

2	6	1	5	3	4	9	7	8
7	5	8	9	2	6	4	3	1
3	4	9	7	1	8	6	2	5
5	9	6	4	8	7	3	1	2
1	8	2	3	6	5	7	4	9
4	7	3	2	9	1	8	5	6
8	2	4	1	7	9	5	6	3
9	1	5	6	4	3	2	8	7
6	3	7	8	5	2	1	9	4

EXERCISE 33 PAGE 71

Respectively:

6	9	3	7	8	4	5	1	2
4	8	7	5	1	2	9	3	6
1	2	5	9	6	3	8	7	4
9	3	2	6	5	1	4	8	7
5	6	8	2	4	7	3	9	1
7	4	1	3	9	8	6	2	5
3	1	9	4	7	5	2	6	8
8	5	6	1	2	9	7	4	3
2	7	4	8	3	6	1	5	9

4	8	3	7	1	9	5	2	6
1	2	7	4	5	6	3	9	8
9	6	5	3	8	2	1	7	4
7	9	8	1	3	4	2	6	5
6	3	2	9	7	5	8	4	1
5	1	4	2	6	8	9	3	7
3	4	1	5	2	7	6	8	9
2	7	6	8	9	1	4	5	3
8	5	9	6	4	3	7	1	2

EXERCISE 34 PAGE 74

The difficult level is only 2 because rules 2.1 and 2.2 are enough to solve this grid:

4	3	5	1	9	2	8	7	6
9	7	2	8	6	5	1	3	4
1	8	6	3	4	7	2	9	5
2	6	8	5	3	4	7	1	9
5	4	1	9	7	8	6	2	3
7	9	3	2	1	6	4	5	8
6	5	4	7	2	3	9	8	1
3	1	7	6	8	9	5	4	2
8	2	9	4	5	1	3	6	7

Appendix B

Solutions

数独 1

7	5	1	2	6	9	3	8	4
6	3	4	1	5	8	9	2	7
8	2	9	3	4	7	6	5	1
2	9	5	4	7	6	1	3	8
1	4	6	8	9	3	5	7	2
3	7	8	5	1	2	4	9	6
4	1	3	7	2	5	8	6	9
9	8	2	6	3	1	7	4	5
5	6	7	9	8	4	2	1	3

数独 2

8	5	2	9	1	3	7	6	4
3	4	1	5	6	7	8	2	9
7	9	6	2	4	8	1	3	5
4	1	7	3	2	6	9	5	8
2	6	8	7	5	9	4	1	3
9	3	5	1	8	4	2	7	6
5	7	4	6	9	2	3	8	1
1	2	9	8	3	5	6	4	7
6	8	3	4	7	1	5	9	2

数独 3

7	9	6	8	3	5	2	4	1
8	2	5	9	1	4	7	6	3
4	3	1	7	6	2	8	9	5
1	6	4	2	8	9	5	3	7
3	5	9	4	7	1	6	8	2
2	7	8	6	5	3	4	1	9
6	1	2	5	9	8	3	7	4
5	8	3	1	4	7	9	2	6
9	4	7	3	2	6	1	5	8

数独 4

7	3	1	6	9	4	2	5	8
8	4	6	2	1	5	3	7	9
2	9	5	7	3	8	1	6	4
9	6	7	8	4	1	5	3	2
1	8	4	3	5	2	6	9	7
5	2	3	9	7	6	4	8	1
3	5	9	4	2	7	8	1	6
6	1	2	5	8	9	7	4	3
4	7	8	1	6	3	9	2	5

数独 5

4	3	8	1	2	7	5	9	6
1	2	9	6	5	8	3	4	7
6	5	7	3	4	9	8	2	1
9	7	3	8	6	2	1	5	4
8	6	4	5	9	1	7	3	2
2	1	5	7	3	4	6	8	9
5	8	2	4	1	6	9	7	3
7	9	6	2	8	3	4	1	5
3	4	1	9	7	5	2	6	8

数独 6

5	4	2	3	8	6	1	7	9
7	6	8	9	1	4	3	2	5
9	1	3	7	5	2	4	6	8
4	8	5	1	7	3	2	9	6
1	3	9	6	2	8	7	5	4
6	2	7	4	9	5	8	1	3
2	5	6	8	3	1	9	4	7
8	9	4	2	6	7	5	3	1
3	7	1	5	4	9	6	8	2

数独 7

1	8	6	3	7	2	5	4	9
7	3	9	4	5	8	6	1	2
5	2	4	1	6	9	3	8	7
6	9	5	2	8	7	4	3	1
4	7	3	5	9	1	8	2	6
2	1	8	6	3	4	7	9	5
8	6	1	7	2	3	9	5	4
3	5	2	9	4	6	1	7	8
9	4	7	8	1	5	2	6	3

数独 8

8	1	7	9	4	2	3	5	6
5	3	4	6	1	8	2	7	9
2	6	9	7	3	5	1	4	8
9	4	2	8	6	3	7	1	5
3	5	1	2	7	9	6	8	4
6	7	8	4	5	1	9	3	2
7	9	5	3	8	6	4	2	1
1	2	3	5	9	4	8	6	7
4	8	6	1	2	7	5	9	3

数独 9

9	8	4	2	6	1	5	7	3
6	2	5	3	9	7	8	4	1
3	7	1	5	8	4	6	2	9
7	5	6	9	4	2	3	1	8
4	1	9	8	7	3	2	6	5
2	3	8	1	5	6	7	9	4
8	6	7	4	1	5	9	3	2
5	4	2	7	3	9	1	8	6
1	9	3	6	2	8	4	5	7

数独 10

3	9	6	1	5	4	8	2	7
5	1	8	9	7	2	6	3	4
7	2	4	3	8	6	9	1	5
9	6	7	8	2	3	4	5	1
2	5	1	7	4	9	3	8	6
4	8	3	5	6	1	7	9	2
6	3	9	4	1	5	2	7	8
8	4	5	2	3	7	1	6	9
1	7	2	6	9	8	5	4	3

数独 11

8	4	5	1	2	9	7	6	3
3	6	1	7	5	4	8	2	9
2	7	9	6	3	8	4	1	5
7	9	2	3	1	5	6	4	8
1	3	6	4	8	7	5	9	2
5	8	4	9	6	2	3	7	1
4	1	7	8	9	3	2	5	6
9	5	3	2	4	6	1	8	7
6	2	8	5	7	1	9	3	4

数独 12

5	9	3	1	2	7	6	4	8
1	7	2	8	6	4	5	3	9
6	8	4	5	3	9	2	1	7
2	3	5	7	9	1	8	6	4
8	1	6	4	5	3	7	9	2
9	4	7	2	8	6	1	5	3
3	5	8	6	4	2	9	7	1
4	2	1	9	7	5	3	8	6
7	6	9	3	1	8	4	2	5

数独 13

7	2	6	9	3	1	5	4	8
9	3	1	8	5	4	2	7	6
8	4	5	7	2	6	1	3	9
1	6	9	2	4	7	3	8	5
3	5	2	6	9	8	7	1	4
4	8	7	3	1	5	6	9	2
6	7	3	4	8	2	9	5	1
5	9	8	1	6	3	4	2	7
2	1	4	5	7	9	8	6	3

数独 14

7	9	4	5	1	8	3	2	6
6	3	1	7	4	2	8	5	9
5	8	2	9	3	6	7	4	1
1	5	3	4	6	9	2	7	8
8	2	9	1	7	5	6	3	4
4	6	7	8	2	3	1	9	5
3	4	5	2	8	1	9	6	7
2	7	8	6	9	4	5	1	3
9	1	6	3	5	7	4	8	2

数独 15

6	4	1	2	9	8	7	5	3
7	2	5	6	3	4	1	9	8
3	9	8	5	1	7	2	4	6
1	6	4	9	2	3	8	7	5
8	5	9	1	7	6	4	3	2
2	7	3	8	4	5	6	1	9
4	8	6	7	5	9	3	2	1
9	1	7	3	6	2	5	8	4
5	3	2	4	8	1	9	6	7

数独 16

3	6	7	2	9	1	4	5	8
9	1	5	8	7	4	2	6	3
8	2	4	6	5	3	9	1	7
4	7	2	3	6	9	1	8	5
1	9	3	5	2	8	6	7	4
6	5	8	1	4	7	3	9	2
2	8	9	4	1	5	7	3	6
5	4	1	7	3	6	8	2	9
7	3	6	9	8	2	5	4	1

数独 17

5	8	2	1	9	7	6	4	3
1	3	7	8	4	6	9	5	2
6	4	9	2	5	3	1	7	8
9	2	4	3	8	5	7	1	6
3	5	1	6	7	4	2	8	9
8	7	6	9	1	2	5	3	4
7	1	8	4	2	9	3	6	5
4	9	3	5	6	1	8	2	7
2	6	5	7	3	8	4	9	1

数独 18

9	8	7	6	4	5	2	1	3
5	3	4	2	1	9	6	8	7
1	6	2	3	7	8	9	4	5
8	4	1	7	5	2	3	6	9
6	9	5	4	3	1	7	2	8
7	2	3	9	8	6	4	5	1
2	1	9	5	6	7	8	3	4
4	5	6	8	9	3	1	7	2
3	7	8	1	2	4	5	9	6

数独 19

4	2	9	8	3	6	7	1	5
7	5	8	4	2	1	3	9	6
6	3	1	9	7	5	4	2	8
5	4	6	1	9	2	8	7	3
8	1	7	3	6	4	2	5	9
3	9	2	7	5	8	1	6	4
2	8	5	6	4	7	9	3	1
9	6	4	2	1	3	5	8	7
1	7	3	5	8	9	6	4	2

数独 20

2	4	5	3	9	1	8	6	7
7	3	6	2	4	8	9	1	5
1	9	8	6	5	7	4	2	3
8	2	3	7	1	4	5	9	6
9	6	1	5	8	2	7	3	4
4	5	7	9	6	3	1	8	2
5	8	9	4	3	6	2	7	1
6	1	2	8	7	5	3	4	9
3	7	4	1	2	9	6	5	8

数独 21

2	9	3	5	7	6	1	8	4
5	8	4	1	9	3	2	7	6
6	1	7	2	4	8	5	9	3
9	4	2	8	5	1	3	6	7
8	3	6	7	2	9	4	1	5
7	5	1	3	6	4	8	2	9
3	7	5	9	1	2	6	4	8
1	6	8	4	3	7	9	5	2
4	2	9	6	8	5	7	3	1

数独 22

9	6	4	3	1	8	5	2	7
2	5	1	9	7	4	6	8	3
8	3	7	5	6	2	4	9	1
7	4	3	1	8	5	2	6	9
5	9	2	6	4	3	1	7	8
1	8	6	7	2	9	3	4	5
6	1	9	2	3	7	8	5	4
3	7	8	4	5	6	9	1	2
4	2	5	8	9	1	7	3	6

数独 23

7	8	3	1	2	6	5	4	9
6	4	9	7	8	5	3	1	2
1	5	2	9	3	4	8	6	7
2	1	8	6	5	9	7	3	4
5	7	6	2	4	3	9	8	1
3	9	4	8	1	7	2	5	6
4	2	5	3	9	1	6	7	8
8	3	7	4	6	2	1	9	5
9	6	1	5	7	8	4	2	3

数独 24

2	7	8	4	9	5	1	6	3
9	4	5	1	6	3	7	8	2
6	3	1	8	2	7	4	9	5
7	6	4	2	5	8	9	3	1
3	5	9	6	1	4	8	2	7
8	1	2	3	7	9	5	4	6
4	2	6	5	8	1	3	7	9
5	8	7	9	3	2	6	1	4
1	9	3	7	4	6	2	5	8

数独 25

3	6	7	4	1	9	2	5	8
5	8	9	2	3	6	4	7	1
4	2	1	7	5	8	3	6	9
9	7	5	6	8	3	1	4	2
2	4	3	1	9	7	6	8	5
8	1	6	5	4	2	9	3	7
1	5	8	9	6	4	7	2	3
7	3	4	8	2	1	5	9	6
6	9	2	3	7	5	8	1	4

数独 26

9	4	2	6	7	8	3	5	1
8	5	7	3	2	1	9	4	6
1	3	6	4	9	5	8	7	2
4	6	3	7	8	2	5	1	9
7	8	1	9	5	6	2	3	4
2	9	5	1	4	3	7	6	8
3	7	4	2	6	9	1	8	5
5	1	9	8	3	4	6	2	7
6	2	8	5	1	7	4	9	3

数独 27

3	9	4	6	7	8	1	2	5
6	7	5	3	2	1	9	8	4
2	1	8	4	9	5	7	3	6
8	2	6	5	1	9	3	4	7
1	4	7	2	3	6	8	5	9
9	5	3	8	4	7	6	1	2
5	8	2	7	6	3	4	9	1
4	6	1	9	8	2	5	7	3
7	3	9	1	5	4	2	6	8

数独 28

1	5	6	4	8	3	7	2	9
7	4	9	2	6	5	8	3	1
8	3	2	1	9	7	6	4	5
3	6	4	9	1	2	5	7	8
9	2	7	3	5	8	1	6	4
5	1	8	6	7	4	3	9	2
4	8	5	7	2	6	9	1	3
6	9	3	5	4	1	2	8	7
2	7	1	8	3	9	4	5	6

数独 29

1	6	2	4	7	3	5	8	9
4	8	3	9	5	1	2	7	6
5	9	7	8	6	2	4	1	3
3	2	1	5	8	4	9	6	7
6	5	9	1	3	7	8	4	2
8	7	4	6	2	9	1	3	5
2	1	5	7	4	6	3	9	8
9	3	6	2	1	8	7	5	4
7	4	8	3	9	5	6	2	1

数独 30

9	2	5	6	8	4	3	1	7
6	7	1	2	5	3	4	8	9
8	3	4	1	7	9	6	2	5
3	9	6	5	4	2	8	7	1
7	1	2	9	3	8	5	6	4
5	4	8	7	6	1	2	9	3
2	8	3	4	9	7	1	5	6
4	6	7	8	1	5	9	3	2
1	5	9	3	2	6	7	4	8

数独 31

2	8	9	5	7	6	4	1	3
3	5	7	4	9	1	6	2	8
6	1	4	3	2	8	5	9	7
5	4	6	9	1	3	8	7	2
7	3	1	8	4	2	9	6	5
8	9	2	6	5	7	1	3	4
9	6	8	7	3	4	2	5	1
1	7	5	2	8	9	3	4	6
4	2	3	1	6	5	7	8	9

数独 32

4	6	8	5	1	9	7	3	2
5	2	1	8	7	3	4	9	6
9	7	3	6	4	2	5	1	8
8	9	2	7	6	1	3	4	5
6	1	7	3	5	4	8	2	9
3	4	5	2	9	8	1	6	7
1	3	6	9	8	5	2	7	4
7	5	4	1	2	6	9	8	3
2	8	9	4	3	7	6	5	1

数独 33

5	1	2	7	4	3	9	6	8
7	4	8	1	9	6	5	2	3
9	6	3	2	5	8	7	1	4
4	8	7	9	6	1	2	3	5
6	9	5	3	2	4	1	8	7
3	2	1	8	7	5	6	4	9
2	5	4	6	3	7	8	9	1
1	3	6	5	8	9	4	7	2
8	7	9	4	1	2	3	5	6

数独 34

1	7	6	4	9	8	5	2	3
9	4	2	3	7	5	8	6	1
5	8	3	2	1	6	4	9	7
8	2	7	6	5	4	3	1	9
3	9	4	7	8	1	2	5	6
6	1	5	9	3	2	7	4	8
7	6	1	5	2	3	9	8	4
4	5	9	8	6	7	1	3	2
2	3	8	1	4	9	6	7	5

数独 35

7	6	4	3	2	8	9	5	1
1	9	2	4	5	6	7	8	3
3	5	8	7	1	9	4	2	6
8	2	5	9	3	7	1	6	4
6	1	9	8	4	5	2	3	7
4	7	3	1	6	2	5	9	8
5	8	7	6	9	1	3	4	2
9	4	6	2	7	3	8	1	5
2	3	1	5	8	4	6	7	9

数独 36

9	6	1	8	4	2	5	7	3
5	2	7	9	6	3	8	4	1
8	4	3	7	5	1	2	6	9
6	5	8	1	2	4	3	9	7
1	9	2	5	3	7	6	8	4
3	7	4	6	9	8	1	2	5
7	3	5	2	8	9	4	1	6
4	8	9	3	1	6	7	5	2
2	1	6	4	7	5	9	3	8

数独 37

3	9	1	8	7	2	4	6	5
5	8	6	1	9	4	3	2	7
7	2	4	3	5	6	9	1	8
9	6	3	5	8	1	7	4	2
2	7	5	4	6	3	1	8	9
4	1	8	7	2	9	5	3	6
8	4	9	2	1	5	6	7	3
6	3	7	9	4	8	2	5	1
1	5	2	6	3	7	8	9	4

数独 38

4	8	1	6	7	5	3	9	2
6	7	2	3	9	1	4	8	5
3	9	5	8	4	2	7	1	6
7	6	3	1	5	8	2	4	9
5	1	4	9	2	7	6	3	8
9	2	8	4	6	3	1	5	7
8	4	6	2	1	9	5	7	3
2	5	9	7	3	4	8	6	1
1	3	7	5	8	6	9	2	4

数独 39

7	2	5	8	1	3	4	6	9
3	8	6	4	7	9	1	5	2
1	9	4	2	6	5	7	3	8
9	6	2	5	8	1	3	4	7
8	7	1	3	4	6	2	9	5
5	4	3	9	2	7	6	8	1
4	5	8	7	3	2	9	1	6
2	1	9	6	5	4	8	7	3
6	3	7	1	9	8	5	2	4

数独 40

9	4	5	6	2	1	3	8	7
2	1	3	5	8	7	9	4	6
7	6	8	4	3	9	2	5	1
4	3	7	9	5	6	8	1	2
1	8	9	3	4	2	6	7	5
6	5	2	7	1	8	4	9	3
8	7	6	1	9	3	5	2	4
5	9	1	2	6	4	7	3	8
3	2	4	8	7	5	1	6	9

数独 41

3	6	1	5	9	4	8	7	2
2	8	7	3	6	1	5	4	9
5	9	4	7	2	8	1	3	6
1	3	9	8	5	2	4	6	7
7	2	8	4	1	6	9	5	3
6	4	5	9	7	3	2	1	8
9	7	3	1	8	5	6	2	4
4	1	6	2	3	9	7	8	5
8	5	2	6	4	7	3	9	1

数独 42

8	5	2	9	1	4	7	6	3
1	7	4	6	3	2	9	8	5
9	6	3	8	5	7	1	2	4
2	4	1	3	8	6	5	7	9
5	8	6	7	9	1	3	4	2
7	3	9	2	4	5	6	1	8
3	2	7	4	6	9	8	5	1
6	9	5	1	2	8	4	3	7
4	1	8	5	7	3	2	9	6

数独 43

1	5	3	8	7	6	2	4	9
6	4	9	1	2	3	7	5	8
2	7	8	5	9	4	6	3	1
9	1	5	2	8	7	3	6	4
4	3	2	6	5	9	1	8	7
8	6	7	3	4	1	5	9	2
5	8	1	9	6	2	4	7	3
7	2	6	4	3	8	9	1	5
3	9	4	7	1	5	8	2	6

数独 44

5	2	1	8	7	6	4	9	3
4	8	9	1	2	3	5	7	6
6	3	7	5	9	4	8	1	2
7	4	3	9	1	5	6	2	8
9	5	8	6	3	2	7	4	1
2	1	6	7	4	8	3	5	9
1	9	4	3	6	7	2	8	5
3	7	5	2	8	9	1	6	4
8	6	2	4	5	1	9	3	7

数独 45

9	2	7	3	8	4	6	5	1
1	3	8	5	6	2	9	4	7
5	4	6	7	9	1	2	3	8
8	7	5	2	1	9	4	6	3
4	6	1	8	5	3	7	2	9
2	9	3	4	7	6	8	1	5
3	1	9	6	2	7	5	8	4
7	8	2	1	4	5	3	9	6
6	5	4	9	3	8	1	7	2

数独 46

9	3	6	2	8	5	1	4	7
2	7	1	3	6	4	9	8	5
5	4	8	1	7	9	6	2	3
1	9	5	6	2	8	7	3	4
3	6	7	5	4	1	2	9	8
8	2	4	7	9	3	5	1	6
4	8	2	9	5	6	3	7	1
6	1	9	8	3	7	4	5	2
7	5	3	4	1	2	8	6	9

数独 47

2	9	1	5	7	4	3	6	8
7	3	8	6	2	9	1	4	5
4	6	5	8	3	1	9	7	2
8	5	4	1	6	3	7	2	9
9	1	7	2	4	5	6	8	3
6	2	3	9	8	7	5	1	4
3	8	2	7	9	6	4	5	1
1	4	6	3	5	2	8	9	7
5	7	9	4	1	8	2	3	6

数独 48

6	2	9	1	7	8	5	3	4
4	8	5	9	2	3	6	7	1
1	3	7	4	5	6	8	2	9
3	7	6	5	1	9	4	8	2
2	9	1	7	8	4	3	6	5
5	4	8	6	3	2	1	9	7
9	5	4	3	6	7	2	1	8
8	1	3	2	9	5	7	4	6
7	6	2	8	4	1	9	5	3

数独 49

2	8	3	4	6	5	7	1	9
5	4	6	1	9	7	2	8	3
9	1	7	8	2	3	4	5	6
3	2	1	5	7	9	8	6	4
6	9	5	2	4	8	3	7	1
8	7	4	3	1	6	5	9	2
7	5	2	6	3	1	9	4	8
4	6	9	7	8	2	1	3	5
1	3	8	9	5	4	6	2	7

数独 50

4	8	5	9	1	6	2	7	3
1	9	3	7	2	8	5	4	6
7	6	2	4	3	5	8	1	9
8	1	4	5	9	7	3	6	2
9	5	6	3	4	2	7	8	1
3	2	7	6	8	1	4	9	5
5	3	9	1	7	4	6	2	8
2	7	1	8	6	3	9	5	4
6	4	8	2	5	9	1	3	7

数独 51

3	5	1	9	2	8	7	4	6
6	2	7	4	1	5	3	9	8
4	8	9	3	6	7	5	2	1
9	1	4	7	3	2	8	6	5
8	7	3	6	5	4	2	1	9
5	6	2	8	9	1	4	3	7
2	9	6	5	7	3	1	8	4
1	4	5	2	8	6	9	7	3
7	3	8	1	4	9	6	5	2

数独 52

8	1	9	3	2	7	6	4	5
6	2	5	1	9	4	8	3	7
7	3	4	5	8	6	2	1	9
1	6	8	2	4	5	7	9	3
9	4	3	6	7	8	5	2	1
5	7	2	9	3	1	4	8	6
3	9	7	4	5	2	1	6	8
4	5	6	8	1	3	9	7	2
2	8	1	7	6	9	3	5	4

数独 53

3	5	6	1	4	9	2	8	7
2	8	4	3	7	5	1	6	9
7	1	9	2	8	6	5	3	4
6	7	3	8	9	2	4	5	1
1	4	5	6	3	7	8	9	2
8	9	2	5	1	4	6	7	3
4	6	7	9	5	1	3	2	8
9	2	8	4	6	3	7	1	5
5	3	1	7	2	8	9	4	6

数独 54

1	2	3	8	7	6	4	5	9
4	7	9	5	3	1	2	6	8
8	6	5	2	4	9	1	7	3
5	4	6	7	9	8	3	2	1
3	9	7	1	2	5	8	4	6
2	1	8	3	6	4	7	9	5
7	8	2	9	5	3	6	1	4
6	5	1	4	8	7	9	3	2
9	3	4	6	1	2	5	8	7

数独 55

5	9	2	4	3	1	7	6	8
3	6	4	7	8	9	5	1	2
8	7	1	2	6	5	3	9	4
2	8	9	3	1	7	6	4	5
1	4	3	8	5	6	2	7	9
6	5	7	9	4	2	8	3	1
4	2	6	5	9	3	1	8	7
7	1	8	6	2	4	9	5	3
9	3	5	1	7	8	4	2	6

数独 56

6	8	3	4	9	5	1	2	7
9	2	5	8	1	7	3	4	6
4	1	7	3	2	6	5	8	9
2	4	6	5	7	1	8	9	3
3	7	1	9	4	8	6	5	2
5	9	8	6	3	2	7	1	4
1	5	4	2	6	3	9	7	8
7	3	9	1	8	4	2	6	5
8	6	2	7	5	9	4	3	1

数独 57

5	9	4	1	6	7	8	2	3
7	2	3	8	4	5	1	9	6
8	6	1	9	3	2	5	4	7
2	4	7	5	1	6	3	8	9
9	3	6	7	2	8	4	1	5
1	5	8	3	9	4	6	7	2
4	1	2	6	7	3	9	5	8
3	8	9	2	5	1	7	6	4
6	7	5	4	8	9	2	3	1

数独 58

5	8	4	6	7	9	2	3	1
9	1	7	2	8	3	4	5	6
3	6	2	5	1	4	9	8	7
8	9	6	1	5	2	7	4	3
4	2	3	9	6	7	8	1	5
1	7	5	3	4	8	6	2	9
7	4	1	8	3	6	5	9	2
6	3	9	4	2	5	1	7	8
2	5	8	7	9	1	3	6	4

数独 59

9	8	6	2	1	7	5	3	4
4	5	7	9	3	6	1	2	8
1	2	3	4	8	5	6	9	7
2	9	8	5	4	3	7	6	1
6	4	1	7	2	8	9	5	3
7	3	5	6	9	1	8	4	2
8	7	2	3	5	9	4	1	6
3	6	9	1	7	4	2	8	5
5	1	4	8	6	2	3	7	9

数独 60

6	2	1	4	9	5	8	3	7
9	8	3	1	7	6	5	4	2
5	7	4	3	8	2	1	6	9
7	6	2	9	3	1	4	8	5
8	1	9	7	5	4	3	2	6
4	3	5	6	2	8	7	9	1
1	4	8	5	6	9	2	7	3
2	9	7	8	1	3	6	5	4
3	5	6	2	4	7	9	1	8

数独 61

6	7	2	5	4	3	8	1	9
9	5	3	1	8	7	6	2	4
4	1	8	9	2	6	7	3	5
7	8	5	2	9	4	1	6	3
1	6	4	3	7	5	2	9	8
2	3	9	6	1	8	5	4	7
8	2	6	7	3	9	4	5	1
5	9	7	4	6	1	3	8	2
3	4	1	8	5	2	9	7	6

数独 62

2	9	1	5	4	7	8	3	6
5	3	7	2	6	8	9	1	4
6	4	8	9	1	3	5	2	7
8	1	4	6	5	9	3	7	2
9	7	6	4	3	2	1	5	8
3	5	2	8	7	1	6	4	9
4	8	9	3	2	5	7	6	1
1	2	5	7	9	6	4	8	3
7	6	3	1	8	4	2	9	5

数独 63

7	6	4	8	3	1	5	2	9
5	1	8	2	6	9	4	7	3
3	9	2	4	7	5	8	1	6
4	8	9	7	5	6	1	3	2
1	3	5	9	4	2	7	6	8
2	7	6	3	1	8	9	4	5
8	4	3	5	2	7	6	9	1
6	5	7	1	9	3	2	8	4
9	2	1	6	8	4	3	5	7

数独 64

5	1	3	6	9	8	4	7	2
7	4	8	2	1	5	6	3	9
9	6	2	3	7	4	5	8	1
4	3	7	9	2	1	8	6	5
1	5	9	8	4	6	3	2	7
8	2	6	7	5	3	1	9	4
2	8	4	5	6	7	9	1	3
3	9	5	1	8	2	7	4	6
6	7	1	4	3	9	2	5	8

数独 65

7	8	9	5	2	3	1	4	6
6	1	5	4	9	7	3	8	2
2	3	4	8	1	6	7	5	9
8	2	3	6	7	5	4	9	1
4	9	6	3	8	1	5	2	7
1	5	7	2	4	9	6	3	8
9	7	2	1	3	4	8	6	5
5	4	1	9	6	8	2	7	3
3	6	8	7	5	2	9	1	4

数独 66

1	4	7	8	6	3	9	5	2
3	6	5	7	9	2	1	4	8
8	9	2	4	5	1	7	6	3
9	7	6	3	2	5	8	1	4
5	8	3	1	4	7	2	9	6
4	2	1	6	8	9	3	7	5
6	1	9	5	3	8	4	2	7
2	3	4	9	7	6	5	8	1
7	5	8	2	1	4	6	3	9

数独 67

6	2	5	3	9	8	7	1	4
4	7	3	2	5	1	6	9	8
8	1	9	7	6	4	2	3	5
2	8	1	6	3	5	4	7	9
5	6	7	8	4	9	3	2	1
9	3	4	1	2	7	5	8	6
1	9	6	4	7	3	8	5	2
3	5	2	9	8	6	1	4	7
7	4	8	5	1	2	9	6	3

数独 68

9	8	5	3	4	7	1	2	6
2	1	7	5	9	6	3	8	4
4	6	3	2	8	1	9	5	7
8	5	4	1	6	9	2	7	3
7	3	1	4	2	8	5	6	9
6	2	9	7	3	5	4	1	8
3	7	6	9	1	2	8	4	5
1	9	8	6	5	4	7	3	2
5	4	2	8	7	3	6	9	1

数独 69

1	9	8	4	2	5	6	7	3
3	7	2	6	1	9	5	8	4
5	6	4	3	8	7	1	9	2
6	5	7	1	4	3	9	2	8
8	3	1	7	9	2	4	5	6
2	4	9	8	5	6	3	1	7
4	2	6	9	7	1	8	3	5
9	8	5	2	3	4	7	6	1
7	1	3	5	6	8	2	4	9

数独 70

6	7	5	4	2	8	1	9	3
1	4	9	5	3	7	6	2	8
3	2	8	1	6	9	4	5	7
4	6	1	2	8	3	9	7	5
5	9	3	7	4	6	2	8	1
2	8	7	9	5	1	3	4	6
7	3	4	6	9	5	8	1	2
8	1	2	3	7	4	5	6	9
9	5	6	8	1	2	7	3	4

数独 71

6	4	2	5	1	7	9	3	8
3	9	7	8	2	6	1	5	4
5	8	1	3	9	4	6	2	7
1	7	3	6	5	2	4	8	9
4	6	9	7	3	8	2	1	5
8	2	5	1	4	9	7	6	3
7	1	6	4	8	3	5	9	2
9	3	4	2	6	5	8	7	1
2	5	8	9	7	1	3	4	6

数独 72

7	9	8	3	4	5	2	6	1
5	3	6	1	7	2	8	4	9
4	2	1	9	8	6	5	7	3
2	6	3	4	1	7	9	5	8
8	7	5	6	2	9	1	3	4
9	1	4	5	3	8	6	2	7
3	4	2	8	5	1	7	9	6
1	5	9	7	6	3	4	8	2
6	8	7	2	9	4	3	1	5

数独 73

8	5	6	3	9	1	2	4	7
2	3	7	6	8	4	5	9	1
9	4	1	2	7	5	8	3	6
7	1	9	8	3	6	4	5	2
4	2	5	7	1	9	6	8	3
3	6	8	4	5	2	1	7	9
5	7	3	1	6	8	9	2	4
1	8	2	9	4	3	7	6	5
6	9	4	5	2	7	3	1	8

数独 74

5	2	7	1	6	3	8	4	9
4	8	3	9	7	5	1	6	2
6	9	1	8	2	4	3	5	7
8	7	9	3	4	1	5	2	6
3	1	5	6	9	2	7	8	4
2	6	4	7	5	8	9	1	3
1	4	8	2	3	7	6	9	5
9	3	2	5	8	6	4	7	1
7	5	6	4	1	9	2	3	8

数独 75

6	4	2	5	1	7	3	8	9
3	9	5	8	6	2	4	1	7
7	1	8	9	3	4	5	6	2
1	7	6	3	2	8	9	4	5
4	8	9	7	5	1	6	2	3
2	5	3	6	4	9	1	7	8
8	3	1	4	7	5	2	9	6
5	2	7	1	9	6	8	3	4
9	6	4	2	8	3	7	5	1

数独 76

9	1	8	5	7	6	3	4	2
4	3	5	9	2	1	7	6	8
6	7	2	3	4	8	9	1	5
7	2	6	8	5	4	1	9	3
5	8	3	7	1	9	6	2	4
1	4	9	2	6	3	5	8	7
3	5	4	6	9	2	8	7	1
8	6	1	4	3	7	2	5	9
2	9	7	1	8	5	4	3	6

数独 77

5	9	8	7	3	1	2	4	6
4	3	6	2	8	9	5	7	1
1	2	7	5	4	6	9	3	8
6	8	2	1	9	3	4	5	7
9	4	3	8	5	7	6	1	2
7	1	5	6	2	4	3	8	9
3	7	1	4	6	2	8	9	5
8	6	4	9	7	5	1	2	3
2	5	9	3	1	8	7	6	4

数独 78

7	6	4	8	3	2	1	5	9
1	8	5	9	4	6	3	7	2
2	3	9	7	5	1	6	4	8
6	9	3	5	1	7	8	2	4
8	4	2	6	9	3	5	1	7
5	7	1	2	8	4	9	6	3
9	2	6	1	7	8	4	3	5
4	5	7	3	6	9	2	8	1
3	1	8	4	2	5	7	9	6

数独 79

2	7	9	3	6	5	8	1	4
1	8	3	7	4	9	5	6	2
4	5	6	8	1	2	9	3	7
8	1	7	4	2	3	6	5	9
6	9	4	5	8	1	2	7	3
3	2	5	6	9	7	4	8	1
7	3	8	2	5	4	1	9	6
9	6	2	1	7	8	3	4	5
5	4	1	9	3	6	7	2	8

数独 80

1	9	2	8	7	4	6	3	5
6	8	5	1	9	3	4	2	7
3	4	7	5	2	6	8	9	1
7	3	1	6	4	8	9	5	2
8	2	6	9	5	1	7	4	3
9	5	4	2	3	7	1	8	6
2	1	8	3	6	9	5	7	4
5	7	9	4	1	2	3	6	8
4	6	3	7	8	5	2	1	9

数独 81

9	5	6	7	1	8	2	4	3
7	2	4	5	3	9	1	6	8
8	3	1	4	6	2	7	5	9
1	6	8	2	5	3	9	7	4
5	4	9	8	7	1	6	3	2
3	7	2	6	9	4	8	1	5
4	1	5	9	8	7	3	2	6
2	9	3	1	4	6	5	8	7
6	8	7	3	2	5	4	9	1

数独 82

8	4	3	5	9	2	6	7	1
6	5	7	8	1	3	9	2	4
9	1	2	6	7	4	3	8	5
4	8	6	7	3	5	1	9	2
3	2	9	1	6	8	5	4	7
1	7	5	4	2	9	8	6	3
7	6	8	2	5	1	4	3	9
5	9	4	3	8	7	2	1	6
2	3	1	9	4	6	7	5	8

数独 83

4	9	6	1	2	8	3	5	7
1	3	8	6	7	5	4	2	9
5	7	2	9	4	3	8	6	1
3	2	5	8	6	1	9	7	4
8	4	1	5	9	7	6	3	2
9	6	7	2	3	4	5	1	8
7	8	9	3	1	6	2	4	5
6	5	4	7	8	2	1	9	3
2	1	3	4	5	9	7	8	6

数独 84

6	4	1	8	9	7	5	2	3
2	8	3	4	6	5	9	1	7
7	5	9	1	3	2	6	8	4
9	6	5	2	4	1	3	7	8
8	3	4	7	5	9	1	6	2
1	7	2	6	8	3	4	9	5
3	2	6	9	7	4	8	5	1
4	1	8	5	2	6	7	3	9
5	9	7	3	1	8	2	4	6

数独 85

4	8	6	5	1	2	7	9	3
7	3	2	8	6	9	4	5	1
9	5	1	3	4	7	6	8	2
6	7	4	1	8	5	3	2	9
8	2	3	9	7	4	5	1	6
5	1	9	2	3	6	8	7	4
1	4	5	7	2	3	9	6	8
3	9	8	6	5	1	2	4	7
2	6	7	4	9	8	1	3	5

数独 86

3	6	2	4	9	8	7	5	1
8	5	7	6	3	1	4	9	2
4	9	1	7	2	5	3	8	6
6	4	3	9	5	7	1	2	8
1	8	9	3	4	2	6	7	5
7	2	5	1	8	6	9	3	4
9	3	8	2	6	4	5	1	7
2	1	6	5	7	9	8	4	3
5	7	4	8	1	3	2	6	9

数独 87

7	1	9	5	8	4	3	6	2
3	6	4	9	2	1	8	7	5
5	2	8	7	6	3	4	9	1
4	8	5	6	7	9	2	1	3
2	3	6	1	4	5	7	8	9
1	9	7	8	3	2	5	4	6
9	4	2	3	1	8	6	5	7
8	7	1	2	5	6	9	3	4
6	5	3	4	9	7	1	2	8

数独 88

4	9	3	2	5	6	1	7	8
7	1	6	9	4	8	3	2	5
2	5	8	7	3	1	9	4	6
5	6	9	3	7	2	8	1	4
3	2	1	8	6	4	5	9	7
8	4	7	5	1	9	6	3	2
9	3	4	6	8	7	2	5	1
6	7	5	1	2	3	4	8	9
1	8	2	4	9	5	7	6	3

数独 89

3	8	1	2	9	5	7	6	4
5	4	6	7	3	1	9	2	8
7	9	2	8	6	4	1	3	5
6	5	7	1	4	3	2	8	9
8	3	4	9	2	7	5	1	6
2	1	9	6	5	8	3	4	7
9	7	3	4	8	2	6	5	1
1	2	8	5	7	6	4	9	3
4	6	5	3	1	9	8	7	2

数独 90

8	3	1	9	6	4	2	7	5
4	9	2	5	7	1	3	8	6
5	6	7	3	2	8	9	4	1
6	8	3	7	4	5	1	9	2
7	4	9	2	1	3	6	5	8
1	2	5	8	9	6	4	3	7
2	5	8	6	3	9	7	1	4
9	1	6	4	8	7	5	2	3
3	7	4	1	5	2	8	6	9

数独 91

5	2	8	4	1	9	3	6	7
9	4	6	7	5	3	2	8	1
1	3	7	6	8	2	5	4	9
2	9	4	5	3	8	7	1	6
6	8	3	2	7	1	4	9	5
7	1	5	9	6	4	8	3	2
8	5	2	3	9	6	1	7	4
3	7	9	1	4	5	6	2	8
4	6	1	8	2	7	9	5	3

数独 92

2	9	4	6	7	5	1	8	3
7	3	6	4	8	1	2	9	5
1	5	8	3	9	2	7	4	6
4	1	9	2	5	8	6	3	7
6	2	7	9	3	4	8	5	1
5	8	3	7	1	6	4	2	9
3	4	2	1	6	9	5	7	8
8	7	1	5	4	3	9	6	2
9	6	5	8	2	7	3	1	4

数独 93

2	8	1	4	9	5	7	3	6
6	3	5	1	2	7	4	9	8
4	7	9	3	8	6	1	5	2
9	4	8	5	1	2	6	7	3
5	6	7	8	4	3	2	1	9
3	1	2	6	7	9	8	4	5
1	5	6	2	3	4	9	8	7
7	2	4	9	5	8	3	6	1
8	9	3	7	6	1	5	2	4

数独 94

3	8	6	2	5	4	9	7	1
7	4	9	8	6	1	2	3	5
1	5	2	3	7	9	6	8	4
8	1	5	4	2	6	3	9	7
9	6	4	7	3	5	1	2	8
2	3	7	1	9	8	4	5	6
4	7	1	9	8	2	5	6	3
5	2	3	6	4	7	8	1	9
6	9	8	5	1	3	7	4	2

数独 95

9	3	5	1	2	6	4	7	8
1	8	7	9	4	3	2	6	5
2	6	4	8	7	5	9	3	1
4	7	6	5	8	1	3	9	2
8	1	3	2	9	4	7	5	6
5	9	2	3	6	7	8	1	4
3	5	8	7	1	2	6	4	9
6	2	1	4	3	9	5	8	7
7	4	9	6	5	8	1	2	3

数独 96

6	5	7	9	4	1	8	2	3
8	1	4	2	5	3	6	7	9
3	2	9	8	7	6	1	4	5
5	4	8	1	9	2	7	3	6
7	9	2	3	6	4	5	1	8
1	3	6	5	8	7	4	9	2
4	8	1	6	2	9	3	5	7
9	6	3	7	1	5	2	8	4
2	7	5	4	3	8	9	6	1

数独 97

1	3	5	9	6	4	2	7	8
2	7	9	8	3	5	4	1	6
6	4	8	2	7	1	5	3	9
9	2	7	4	5	3	8	6	1
4	6	1	7	9	8	3	2	5
5	8	3	6	1	2	9	4	7
3	1	4	5	8	6	7	9	2
7	5	2	1	4	9	6	8	3
8	9	6	3	2	7	1	5	4

数独 98

3	1	2	4	7	5	6	9	8
9	5	8	2	6	3	7	4	1
7	6	4	9	1	8	3	5	2
5	8	6	1	3	4	9	2	7
1	3	9	7	5	2	4	8	6
4	2	7	6	8	9	1	3	5
2	9	1	8	4	6	5	7	3
6	4	5	3	2	7	8	1	9
8	7	3	5	9	1	2	6	4

数独 99

3	9	1	7	8	5	6	4	2
7	6	4	3	9	2	8	1	5
5	8	2	1	6	4	3	7	9
6	4	7	9	5	3	1	2	8
1	3	8	4	2	7	9	5	6
2	5	9	6	1	8	7	3	4
9	2	5	8	7	1	4	6	3
8	7	3	2	4	6	5	9	1
4	1	6	5	3	9	2	8	7

数独 100

6	7	2	4	5	9	3	8	1
3	5	1	8	6	7	4	9	2
4	8	9	1	3	2	7	6	5
8	6	7	5	1	3	9	2	4
5	9	4	2	8	6	1	3	7
1	2	3	7	9	4	6	5	8
7	4	6	9	2	8	5	1	3
9	1	8	3	7	5	2	4	6
2	3	5	6	4	1	8	7	9

数独 101

3	6	5	9	8	1	7	2	4
7	2	1	5	3	4	6	9	8
8	9	4	6	2	7	1	3	5
5	4	7	3	6	9	2	8	1
6	1	2	7	4	8	9	5	3
9	3	8	2	1	5	4	6	7
4	7	9	8	5	6	3	1	2
1	8	3	4	9	2	5	7	6
2	5	6	1	7	3	8	4	9

数独 102

2	4	3	6	1	9	7	8	5
1	5	6	7	8	4	2	3	9
7	9	8	5	3	2	6	4	1
5	3	7	9	6	1	8	2	4
4	6	1	2	7	8	9	5	3
9	8	2	3	4	5	1	6	7
8	7	4	1	2	3	5	9	6
3	1	9	8	5	6	4	7	2
6	2	5	4	9	7	3	1	8

数独 103

5	9	6	8	7	1	3	4	2
7	2	8	4	9	3	1	6	5
4	3	1	6	5	2	9	7	8
9	1	5	2	6	8	7	3	4
2	8	7	3	4	9	6	5	1
3	6	4	7	1	5	2	8	9
8	5	3	9	2	7	4	1	6
1	4	2	5	3	6	8	9	7
6	7	9	1	8	4	5	2	3

数独 104

1	8	6	5	9	3	2	4	7
9	4	3	2	8	7	1	6	5
7	2	5	4	6	1	9	8	3
6	5	4	7	1	8	3	2	9
2	1	8	3	4	9	5	7	6
3	9	7	6	5	2	4	1	8
5	6	9	8	2	4	7	3	1
4	3	1	9	7	6	8	5	2
8	7	2	1	3	5	6	9	4

数独 105

4	5	6	8	7	9	3	2	1
7	1	2	4	3	5	8	6	9
9	8	3	2	6	1	7	4	5
8	3	7	1	9	4	2	5	6
2	6	4	3	5	8	9	1	7
1	9	5	6	2	7	4	3	8
3	7	1	5	8	2	6	9	4
6	4	8	9	1	3	5	7	2
5	2	9	7	4	6	1	8	3

数独 106

3	1	9	4	6	8	2	7	5
6	5	2	9	7	1	8	3	4
7	4	8	2	3	5	9	1	6
2	9	1	7	5	4	6	8	3
5	3	4	8	9	6	1	2	7
8	6	7	3	1	2	5	4	9
4	2	3	6	8	9	7	5	1
1	7	6	5	2	3	4	9	8
9	8	5	1	4	7	3	6	2

数独 107

6	9	7	3	8	5	4	2	1
2	3	5	1	6	4	8	7	9
8	4	1	7	2	9	5	3	6
1	7	6	2	3	8	9	4	5
3	2	9	4	5	1	7	6	8
5	8	4	9	7	6	2	1	3
9	1	3	8	4	7	6	5	2
4	5	2	6	9	3	1	8	7
7	6	8	5	1	2	3	9	4

数独 108

6	9	1	4	8	5	7	3	2
8	7	5	2	1	3	4	6	9
4	3	2	9	6	7	8	5	1
2	5	7	1	9	8	3	4	6
3	8	6	7	4	2	1	9	5
1	4	9	3	5	6	2	7	8
9	2	8	6	7	4	5	1	3
7	1	3	5	2	9	6	8	4
5	6	4	8	3	1	9	2	7

数独 109

4	5	9	3	6	1	2	7	8
6	7	1	8	4	2	9	3	5
2	8	3	7	5	9	1	4	6
1	6	8	5	3	7	4	2	9
5	2	7	1	9	4	6	8	3
3	9	4	2	8	6	5	1	7
9	1	6	4	7	8	3	5	2
8	4	5	6	2	3	7	9	1
7	3	2	9	1	5	8	6	4

数独 110

1	7	4	2	6	8	9	3	5
9	5	8	4	7	3	6	1	2
6	2	3	9	1	5	8	4	7
5	4	9	7	3	1	2	6	8
2	3	6	8	9	4	5	7	1
7	8	1	5	2	6	4	9	3
8	1	7	6	5	9	3	2	4
3	9	5	1	4	2	7	8	6
4	6	2	3	8	7	1	5	9

数独 111

6	7	2	1	4	9	8	3	5
3	9	5	8	2	6	1	4	7
4	8	1	7	5	3	6	2	9
5	2	3	6	1	8	9	7	4
7	4	6	5	9	2	3	1	8
9	1	8	3	7	4	5	6	2
2	3	4	9	6	5	7	8	1
8	5	7	4	3	1	2	9	6
1	6	9	2	8	7	4	5	3

数独 112

7	8	3	9	5	6	2	1	4
9	6	1	4	2	7	3	5	8
5	2	4	3	1	8	7	9	6
4	9	8	1	6	3	5	2	7
3	1	2	7	8	5	6	4	9
6	5	7	2	4	9	1	8	3
8	3	9	5	7	1	4	6	2
2	7	5	6	9	4	8	3	1
1	4	6	8	3	2	9	7	5

数独 113

5	1	9	8	4	6	2	3	7
7	8	3	2	9	1	4	6	5
2	6	4	5	3	7	8	1	9
4	2	1	9	7	5	6	8	3
3	5	8	6	2	4	9	7	1
9	7	6	1	8	3	5	2	4
1	9	2	7	5	8	3	4	6
8	4	7	3	6	9	1	5	2
6	3	5	4	1	2	7	9	8

数独 114

5	7	1	8	2	3	6	9	4
8	6	4	5	1	9	2	3	7
2	3	9	6	4	7	1	8	5
4	9	2	3	7	6	8	5	1
6	8	3	1	5	4	9	7	2
1	5	7	9	8	2	4	6	3
9	4	6	7	3	1	5	2	8
7	1	8	2	9	5	3	4	6
3	2	5	4	6	8	7	1	9

数独 115

4	6	9	2	3	1	8	7	5
7	3	5	8	6	4	1	9	2
1	8	2	5	9	7	3	6	4
3	1	6	4	5	2	9	8	7
8	2	4	9	7	3	5	1	6
5	9	7	1	8	6	2	4	3
9	7	8	3	4	5	6	2	1
2	4	3	6	1	8	7	5	9
6	5	1	7	2	9	4	3	8

数独 116

9	3	7	2	6	1	8	5	4
6	1	8	5	3	4	9	2	7
2	5	4	7	9	8	6	3	1
8	9	5	6	4	3	1	7	2
3	6	2	1	5	7	4	9	8
7	4	1	8	2	9	5	6	3
4	8	3	9	7	5	2	1	6
5	7	6	4	1	2	3	8	9
1	2	9	3	8	6	7	4	5

数独 117

6	1	3	7	9	2	4	8	5
2	5	4	6	3	8	7	9	1
9	8	7	5	1	4	2	3	6
7	6	5	9	4	3	1	2	8
8	3	2	1	5	7	6	4	9
1	4	9	2	8	6	5	7	3
5	7	8	4	6	9	3	1	2
3	2	6	8	7	1	9	5	4
4	9	1	3	2	5	8	6	7

数独 118

9	2	6	5	8	7	4	3	1
5	7	4	2	1	3	9	6	8
1	8	3	6	9	4	2	5	7
3	9	5	8	2	1	7	4	6
7	6	1	4	5	9	8	2	3
8	4	2	3	7	6	1	9	5
4	5	8	1	3	2	6	7	9
6	1	7	9	4	5	3	8	2
2	3	9	7	6	8	5	1	4

数独 119

9	7	5	4	1	6	2	3	8
1	3	4	2	5	8	6	7	9
2	8	6	3	7	9	5	4	1
4	5	8	6	9	2	7	1	3
7	6	1	5	3	4	8	9	2
3	9	2	7	8	1	4	5	6
6	4	9	1	2	7	3	8	5
8	2	3	9	4	5	1	6	7
5	1	7	8	6	3	9	2	4

数独 120

7	8	6	2	3	4	5	1	9
2	3	1	7	9	5	6	4	8
9	5	4	6	1	8	3	2	7
4	7	8	3	2	6	9	5	1
6	9	2	8	5	1	4	7	3
3	1	5	4	7	9	8	6	2
1	2	9	5	6	3	7	8	4
8	6	3	1	4	7	2	9	5
5	4	7	9	8	2	1	3	6

数独 121

8	2	9	1	3	7	4	6	5
4	1	5	6	8	2	9	7	3
7	6	3	5	9	4	1	2	8
3	7	1	8	4	6	5	9	2
6	8	2	9	1	5	7	3	4
9	5	4	7	2	3	8	1	6
5	4	7	3	6	9	2	8	1
2	3	8	4	7	1	6	5	9
1	9	6	2	5	8	3	4	7

数独 122

6	3	8	7	2	5	4	1	9
1	2	5	9	4	6	7	3	8
7	9	4	8	3	1	5	2	6
8	7	3	4	9	2	1	6	5
9	1	6	3	5	7	8	4	2
5	4	2	1	6	8	3	9	7
2	8	7	6	1	4	9	5	3
4	5	9	2	8	3	6	7	1
3	6	1	5	7	9	2	8	4

数独 123

3	6	7	2	5	4	9	1	8
8	2	9	3	1	6	5	4	7
1	4	5	8	9	7	2	6	3
9	1	6	4	8	2	3	7	5
7	8	3	5	6	1	4	9	2
2	5	4	7	3	9	1	8	6
6	7	2	1	4	3	8	5	9
4	9	8	6	2	5	7	3	1
5	3	1	9	7	8	6	2	4

数独 124

6	9	8	3	7	1	4	5	2
4	1	5	9	2	8	6	7	3
2	3	7	5	4	6	9	1	8
5	7	2	4	6	3	1	8	9
9	8	3	2	1	7	5	6	4
1	4	6	8	9	5	3	2	7
8	5	4	6	3	2	7	9	1
7	6	9	1	8	4	2	3	5
3	2	1	7	5	9	8	4	6

数独

125

1	4	6	7	9	2	3	8	5
5	2	8	1	3	4	6	9	7
9	3	7	6	5	8	2	4	1
3	8	1	4	6	9	5	7	2
2	6	5	8	1	7	9	3	4
7	9	4	5	2	3	1	6	8
8	5	3	9	4	1	7	2	6
4	1	9	2	7	6	8	5	3
6	7	2	3	8	5	4	1	9

This page intentionally left blank

Bibliography

[APP 89] APPEL K., HAKEN W., *Every Planar Map is Four Colorable*, vol. 98 of *Contemporary Mathematics*, American Mathematical Society, Providence, RI, 1989.

[BAL 99] BALAKRISHNAN R., RANGANATHAN K., *A Textbook of Graph Theory*, Springer, 1999.

[BOI 93] BOIZUMAULT P., *The Implementation of Prolog*, Princeton Series in Computer Science, Princeton University Press, 1993.

[BRA 86] BRATKO I., *Prolog Programming for Artificial Intelligence*, International Computer Science Series, Addison-Wesley, 1986.

[DEB 01] DEBRUYNE R., BESSIÈRE C., "Domain Filtering Consistencies", *Journal of Artificial Intelligence Research*, vol. 14, p. 205–230, May 2001.

[GAR 79] GAREY M. R., JOHNSON D. S., *Computers and Intractability – A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., New York, 1979.

[RÉG 03] RÉGIN J.-C., "*Constraints and Integer Programming combined*", Chapter Global Constraints and Filtering Algorithms, Kluwer, 2003.

[ROS 06] ROSSI F., VAN BEEK P., WALSH T., Eds., *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, Elsevier Science Publishers, Amsterdam, The Netherlands, 2006.

[STE 86] STERLING L., SHAPIRO E., *The Art of Prolog*, Advanced Programming Techniques, MIT Press, 1986.

This page intentionally left blank

Index

Symbols

$\leftarrow$ 26
$\neq$ 26
$\nleftarrow$ 26

A

acp 65
arc-consistency 66

B

backtracking 59
band 12
block 11
bottom-up generation 76

C

candidate 29
candidate column (rule) 31
candidate lines (rule) 30
candidate rows (rule) 31
cell 11
column 11
complexity 16
constraint 65
    programming 65
    solver 65
contradiction 55
coordinate system 12
counting 18
    different grids 18
    non-equivalent grids 18

D

declarativity 61
different (grid) 18
difficult 43
difficulty rating 74
disjunctive construction (technique) 54
domain reduction 65
duality 44

E

easy 29
enumeration 67
equivalent (grid) 18
expert 52

F

filtering 65
First Order logics 60

G H I J K

generation
    bottom-up 76
    top-down 75
given 11
graph 17
graph coloring 17
grid
    different 18

equivalent 18
irreducible 20
minimal 20
hidden pair (rule) 41
hidden tuple (rule) 42
hierarchy (rules) 73
irreducible (grid) 20
kakuro 21
killer sudoku 21

L

latin square 13
line 26
logic programming 59

M

mark 29
matching 44
medium 34
minimal (grid) 20
multiple lines (rule) 33

N O

naked triple (rule) 38
naked tuple (rule) 39
nampure 13
number place 13
officer problem 12
orthogonal 49

P

possible 26
predicate 60
propagation 66

Q

query 61

R

rating (difficulty) 74
recursivity 63
reductio ad absurdum (technique) 55
region 26
region partitioning 44
row 11
rule
candidate column 31
candidate lines 30
candidate rows 31
hidden pair 41
hidden tuple 42
multiple lines 33
naked pair 37
naked triple 38
naked tuple 39
single candidate 28
single position 27
Swordfish 52
XWing 50
rules (game) 11
rules hierarchy 73

S

satisfiability test 65
single candidate (rule) 28
single position (rule) 27
stack 12
sudoku (constraint programming) 67
sudoku (PROLOG program) 63
Swordfish (rule) 52

T U

technique
disjunctive construction 54
reductio ad absurdum 55
top-down generation 75

V W

valid 11
variable 65
variants 20
very difficult 52
very easy 28
`what` 26
`where` 26

X Y Z

XWing (rule) 50
zone 26