BERT

Base Equation Resolution Theory

Sudoku Solving and Scoring

Bernd Karl Rennhak

Version 0.2

Discussion Paper
Comments, Ideas, Corrections are welcome!

mailto: logel@logelium.de

CONTENTS

1

PREFACE

1.1 ABOUT SUDOKU

It still remains a mystery to me why a mathematical problem that requires a reasonable amount of logic to solve attracts millions of people around the globe.

Definition 1.1. SUDOKU PUZZLE STATEMENT

Fill a 9 by 9 square that contains already some given numbers in a way, that every row, column and box contains all numbers from 1 to 9.

This well known problem statement says *what* a correctly filled Sudoku grid looks like. It says nothing about *how* such a grid should be obtained.

1.2 QUESTIONS AND SOME ANSWERS

This document intends to present a view on Sudoku solving that is different from the traditional and well known concepts. It starts from the very basics and answers the following questions.

- What is a solution ?
- What is a resolution path ?
- What are the steps of a resolution path ?
- How to compare resolution paths ?

Although these questions seem very simple, there is no consensus about the answers in the Sudoku community. Especially the last is the most important, because "simplicity" is only an informal or subjective matter unless there is a well defined measuring method. A consensus would require an absolute answer to these questions. This is asking for too much. This paper introduces a Sudoku theory (BERT) based on very few principles in a consistent way. So the answers to the above questions make only sense within the theory framework. Although a principle based theory inevitably comes with some limitations, the solving scope is large. There is no unsolvable Sudoku puzzle in BERT - afaik.

It is more than fair to clearly mention what this document is *not* intended for or concerned with.

- No new tools or support for improved manual solving will be introduced. There are already enough books and web pages published for this purpose.
- It is *not* intended to give methods to *find* eliminations in the first place, although there are aspects that may inspire some new ideas. The results of existing solvers are used instead.
- It is *not* in the scope of this document, to present a concept for an improved solver. But the results presented show that there is a lot of room for improvements.
- There is no new concept to generate puzzles.

This document addresses a reader already familiar with advanced Sudoku solving concepts that are capable of solving "hard" Sudoku puzzles. These are usually too difficult for humans and require computer aided procedures. Despite starting from elementary definitions, understanding the document requires knowledge of the standard solving methods (*singles*, *pairs*, *triples*, *wings* ...), more sophisticated methods (*fish patterns*, *chains*, *coloring*, *ALS*, *nice loops*, ...) and of recursion techniques. The concepts of BERT are especially useful for "hard" puzzles, but of course they work for all the others too.

Moreover the solving strategies developed by DENIS BERTHIER (DB)[1] using *nrczt-chains*, *whips*, *braids*, etc are referenced frequently. The concepts that ALAN BARKER (AB)[2] used with his solver are also important. Both had significant impact on my ideas. Important work on Sudoku generation and rating was done by GLENN FOWLER (GSF)[3]. I do not mention all the others, who brought ideas and inspiration into the Sudoku discussion. Although the discussion intensity has decreased significantly during the last years, there are still interesting entries in THE NEW SUDOKU PLAYERS FORUM (SPF)[4].

The terminology of traditional Sudoku solving can be found in SUDOPEDIA [5]. This document tries to avoid conflicts with terms defined there as much as possible.

1.3 GENERAL REMARKS

1.3.1 *Traditional Sudoku Solving*

The phrase *traditional* will be always used as a reference to the various methods and logical constructions used with manual solving. This designation is not meant to devalue their importance. There is also no precise definition, but rather an ongoing discussion. Traditional solving methods are merely a toolbox of unrelated and at the most partially ordered methods. The method set is also subject to personal preferences.

1.3.2 *Finite Infinity*

There is a common mathematical joke that all finite problems are trivial, because they can be proved by checking all possible combinations always. Although this is very true, this truth is no help when it comes to solve even moderate large problems practically. The number of combinations is simply by far too large to exhaust, even with computer support. This may be hard to accept for some people believing that computers can do anything. Sudoku is a perfect example that this is untrue. Most of the statements that begin with "All Sudoku (or something related) have the property of ..." are theoretically provable, but practically not.

Often a lot of work was invested to find counter-examples of such statements without finding one. So the statement is proven neither true or false, but this unsatisfactory result has still some value. The more moderate statement "All Sudoku of a large list including very hard and complex problems have the property of ..." is appropriate for such cases.

In this document we will use the abbreviation "All(?) Sudoku ...", because this situation occurs quite often.

[1] DENIS BERTHIER www.carva.org/denis.berthier
[2] ALAN BARKER www.sudokuone.com
[3] GLENN FOWLER gsf.cococlyde.org
[4] THE NEW SUDOKU PLAYERS FORUM forum.enjoysudoku.com
[5] SUDOPEDIA sudopedia.enjoysudoku.com/Terminology.html

1.4 CURRENT STATUS OF SOME ISSUES

1.4.1 *Severity Rating*

There are a number of concepts for Sudoku that try to measure the severity of a Sudoku puzzle. All of these concepts are based on properties of the most complex elimination pattern only, afaik. The calculated complexity value is slightly different for the various types of ratings, but mainly depends on the number of true candidates that can exist in an elimination pattern. These ratings are reflecting the intuitive estimation of severity to find a solution more or less.

But we cannot ignore that something is missing, if all but one resolution step is ignored when calculating the severity. Solution paths usually consists of several hard elimination steps.

1.4.2 *Resolution Path Scoring*

If we talk about severity or complexity of Sudoku, one should not confuse the complexity of the process finding a solution and the complexity of the solution path itself. Maybe it's more clear to explain this for a problem from a completely different domain. If a navigation computes a path on a street map, we easily see that the effort of computing the path and the effort to drive the path is not the same thing. Despite of that we can determine the quality of the path in terms of length, estimated time, fuel consumption, etc perfectly without knowing how the navigation was computed. From this perspective Sudoku is no different. We want to assess the quality of a resolution path without using the information what rules or strategies are used to find it. Up to now there seem to be no convincing concept of comparing resolution paths of the same Sudoku problem. Although the claim "my eliminations are simpler" is frequently used in many discussions, a substantial definition of "simplicity" is painfully missing. This issue among others is addressed in this document.

What are the main obstacles?

- There is no generic definition of "elimination rule".
- There is no common unit of measurement.
- The discussion of "ordering the rules" led to nothing.
- Elimination methods may clear many target candidates.
- Similar patterns may share significant parts.

At least one thing should be clear: A numerical value for the complexity of a resolution path should depend on all steps, not just the severest one.

1.4.3 *Solvers*

There are some Sudoku solver of reasonable quality that should be mentioned explicitly.

- *Sudoku Explainer* by GLENN FOWLER
  Seems to solve any puzzle. Uses lengthy dymanic forcing chains and dynamic contradiction chains for hard puzzle.
  Does not use large matrix (base-cover) patterns and no exocet patterns.
  Works step-by-step and offers alternatives.
- *Xsudo* by ALAN BARKER
  Fails on very hard puzzzles. Has good performance and nice graphics.

Does not use or find large matrix or exocet patterns, avoids overlapping bases.[6]
Generates one path only. Allows resolution state editing.

- *PBCS* Project by DENIS BERTHIER
  Uses a special set of pattern constructions (nrczt-chains,whips,braids) and explores the solving limits of these patterns.
  So it does not solve all puzzle by design.

- many other that automate methods of manual solving.
  All these solvers have their limit, when all predefined methods are exausted.
  Then they switch to recursion (brute force) or fail.

All of them give litte to none explanation, why a particular step was chosen in a situation.

1.4.4 *Open Question*

The very simple Sudoku

`000609000001000700020000090800050007700040006400020008030000020005000300000408000`

is solved by 14 *box-line interactions* and *singles* only. Alternatively there is a sequence with one *hidden pair* (numbers 48 in box 7*) and* 3 *box-line interactions*. This leaves us with the question how to trade 11 *box-line* eliminations against 1 *pair*. Without a consistent framework that can deal with wide-ranging kinds of resolution paths this question is impossible to answer. The simplistic approach to define *box-line* simpler than *pair* in an a-priori manner has to be rejected. Although this example is none of "hard" Sudoku puzzles, it raises difficult questions already. In a way it demonstrates the difference between local and global optimization.

The proposal for a resolution path complexity requires some theory, terminology and some preparations. These are explained in the following chapters.

[6] Terms used here will be explained in later chapters.

2

FOUNDATION

This chapter contains some basic definitions and terminology and considerations that will be useful later.

2.1 SOLUTION

The *solution* is concerned with the fact that the final grid is correct, nothing else. If the grid is filled with numbers and the conditions of puzzle statement 1.1 are met, it's a valid solution. It is not required to reveal how the solution is obtained. So if you assume uniqueness or assume a backdoor or try T&E with some candidates or look-up solutions in a table or snoop in the back pages of the Sudoku book, it's all OK. A solution is not less valid if produced by dubious or even by wrong methods.

If the Sudoku problem statement is expressed with equivalent math models, each model comes with an own version of solution description.

2.2 RESOLUTION THEORY

KEY POINT *What is the value of a resolution theory ?*

Any Sudoku problem can be solved with an algorithm that recursively sets numbers in the grid until a solution is found or a contradiction to one of the rules is met. If the recursion is also combined with *preferring bi-value cells and single eliminations*[1] and possibly other easy to spot logical methods, such recursion has low depth and is fairly efficient. So there seem to be little need for other solution methods unless there is a secondary motivation. The recursion method also detects for an arbitrary Sudoku grid, if there is a unique solution or if a hidden contradiction allows no solution. This and any other algorithm with the same capability is called *universal solver* in the following. Such solvers a necessary to perform uniqueness checks on Sudoku problems.

The term *resolution theory* was brought up by DB and defines the *"how"* part of a solution. This part is the *resolution path* consisting of *resolution steps*. Because the resolution steps depend on each other, a *resolution path* is a graph where the *resolution steps* are edges and the the nodes are *resolution states*. This directed graph reflects the dependencies of *resolution steps*.

2.2.1 *Traditional*

Traditional solving uses a toolbox of methods. Although there is a universal method to solve any Sudoku, there are a number of additional requirements how a solution should be reached. Traditional solving stems from manual solving and does not see itself as resolution theory, but instead tries to declare "right" and "wrong" ways for solving. The following arguments to exclude recursions or back-doors are found frequently.

1. use only "pure logic"
2. "guessing" is not allowed
3. steps must be "as simple as possible"

There is no pure or impure logic, there is only correct and incorrect logic.

[1] *Assumed to be well known.*

4. use only "constructive methods"
5. the path must be "easy to explain"

The first one is obscure, but the others have some relevance. Before discussing the remaining requirements, it should be clear that any such requirement imposes restrictions on the resolution. These are not part of the original Sudoku puzzle statement. So why should we burden the solving with such additional requirements? There should be a *very* good reason. The problem here is not the restrictions itself but the missing goal and motivation for it.

A negative approach by excluding unwanted methods also invites difficulties. Long forum discussions about "what exactly is guessing" is only one example.

Traditional solving is *not* universal unless a method is added to the toolbox that is already universal on its own.

2.2.2 *Property View*

A decent resolution theory should have a clear an unambiguous definition of its resolution steps. This approach also keeps some methods out, but its a more constructive way to achieve this. There will be always someone shouting "my favorite method is not included", but this is missing the point. The importance of a well defined resolution theory lies in the fact that results are comparable. Common properties of the results are a prerequisite for comparing. So a resolution theory assigns properties to steps or resolution paths, but these are relative to the resolution theory context. Comparing to properties of other resolution theories makes no sense usually.

So the main purpose of a resolution theory is to define comparable properties to objects of that theory.

A resolution theory is not meant to be universal from the start. It is no error that some Sudoku puzzles are not solvable. If the scope of solvable puzzles inside a theory is too small, this theory has no relevance of course.

2.2.3 *BERT Intro*

The following chapters present a resolution theory that aims at assigning a numerical complexity value to each resolution step and finally to the whole resolution path. All necessary solving restrictions of BERT are devoted to this goal.

There are two main parts. The first defines the scope by defining the resolution steps. These are the building blocks used for BERT resolutions. The second part defines a metric on these steps to make BERT resolution parts comparable. This does not exclude possible alternative metrics on the steps, but is no issue here.

2.3 THE MATH MODEL

KEY POINT *Transformation of the informal Sudoku description into a formal one.*

The original definition of Sudoku is informal. There are different ways to convert Sudoku problems into an equivalent abstract mathematical model. These models usually work with *candidates* representing one of the possible symbols in a cell of the grid. Any cell of the grid has nine candidates, so the complete empty Sudoku grid contains $9 * 9 * 9 = 729$ candidates.

2.3.1 *Boolean Model*

The most obvious modeling is mapping the two possible candidate values into Boolean values and apply Boolean algebra methods to determine all candidate values to solve the Sudoku. This is mostly done by applying "methods" that bundle some logical implications. The main drawback of this model gets evident when formulating the equation e.g. for a row. Only one of the $n = 9$ candidates $d_1, \ldots, d_n$ in that row can be $TRUE$. So a description with Boolean variables needs an OR-equation $d_1 \vee \ldots \vee d_n = TRUE$ and additionally $n * (n-1)/2$ equations with pairwise NAND equations $\forall_{i \neq j} \neg d_i \wedge \neg d_j = TRUE$.
So implications with this model are easy with bi-value relations only. The model works good with *alternating implication chains* or similar constructions. With complex networks the calculations tend to get cumbersome and unpleasant.

2.3.2 *CSP Model*

The modeling used by Denis Berthier relies on so-called 2D-variables that correspond to cells, rows, columns and boxes. There are 324 of such 2D-variables and their solution value is the index of the true candidate. Each candidate connects the values of the four crossing 2D-variables with an appropriate equation. This model views a Sudoku problem as a *constraint satisfaction problem* in the first place. It is very special and not used elsewhere.

For details refer to the personal web pages for various information.

2.3.3 *Arithmetic Model*

The idea to map Boolean values into integers is not new but rarely used for Sudoku solving. Such approach might be surprising at first sight, because the only relevant integers are 0 for invalid and 1 for valid candidates. But the application of this idea to Sudoku problems will make the calculation of complex elimination networks easier to understand. Other advantages of the *arithmetic model* will become obvious later.

It is the model of choice in BERT. The following definitions are the basics of that model.

Definition 2.1. Candidate
The index set $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ contains the Sudoku symbols.
The candidate representing the number n in the cell with row index r and column index c is denoted by d_{nrc}, *where* $n, r, c \in X$
A candidate is an integer variable that has two possible values: *0* or *1*.
The set D contains all candidates.

Remark 2.2. It is important to note that candidate variables are *integer* values. Usually candidates are used only as long as their value is unknown, but in the context of this model we keep the candidate variables all the time regardless of their status.

Remark 2.3. The members of the index set X are actually symbols with no meaning other than identity. Any other symbol set will do.

Then we define equations to be satisfied for the candidates:

Definition 2.4. NATIVE BASE EQUATION of Cell, Row, Column, Box

Cell at row $r \in X$ and column $c \in X$

$$N_{rc} = \sum_{n \in X} d_{nrc} = 1$$

Row $r \in X$ of number $n \in X$:

$$R_{rc} = \sum_{c \in X} d_{nrc} = 1$$

Column $c \in X$ of number $n \in X$:

$$C_{nc} = \sum_{r \in X} d_{nrc} = 1$$

Box $b \in X$ of number $n \in X$:

$$B_{nb} = \sum_{z \in X} d_{nbz} = 1$$

The function $(b, z) = f(r, c)$ must be a one-to-one mapping of $X \times X$ into itself.
The set NBE contains all *native base equations*.

Remark 2.5. The mapping function for the boxes can be assumed to define the traditional 3*3 sub-grids, although this is not required for the subsequent considerations. Boxes may have any shape, but none should match a whole row or column.

The definitions for *row*, *column* and *box* are clearly direct translations from the Sudoku puzzle statement 1.1. The *cell* definition is different because cells are not mentioned verbatim. The phrase "fill the grid" is interpreted such that each position in the grid contains exactly one number, not two or more, and no cell is empty. Although this sounds rather hair-splitting, it is necessary for an exact and unambiguous definition.

Native base equations (nbe) are defined by building the *arithmetical sum* of it's candidate variables. An empty Sudoku grid contains 81 *native base equations* for each set of cells, rows, columns and boxes. So there is a total number of 324. It is an important note that the equations for the four different kinds are structurally equal. The definitions 2.1 and 2.4 allow a formal re-definition of a Sudoku solution in the context of the *arithmetic model* in a very simple and straightforward way.

Definition 2.6. SOLUTION
A *given* is a candidate of the initial puzzle with value 1 .
The set $G \subset D$ contains the givens of a Sudoku problem.
The set D is a solution, if all candidates of D have a definite value and the expression $\forall g \in G\{g = 1\} \wedge \forall nbe \in NBE\{nbe = 1\}$ is satisfied.

Remark 2.7. Some analysis of Sudoku problems show that the solution expression is (always?) over-defined. Some of the *nbe* are redundant even when the problem has a unique solution. The redundancy of the puzzle statement is investigated in chapter chapter 11. It is also the cause of some special effects that occur in large elimination patterns.

Conjecture 2.8. *All(?)*[2] *Sudoku with a unique solution need at least 17 givens.*

This conjecture was or is still a subject of intense investigations.

[2] see section 1.3.2 explaining "All(?)"

The main idea of BERT is to treat Sudoku as system of 324 linear Diophantine equations. Combined with the value restrictions of the candidates such a system may have one, many or no solutions naturally. Sudoku with a unique solution are of course the most interesting ones. Uniqueness is a delicate issue that sparked some disputes in the Sudoku community and is discussed later.

The central theme of this document is about resolution paths where all steps are based on arithmetic operations on equations of that system. Only such resolution paths will be accepted. We will see that a few – but not many – types of operations are sufficient to describe BERT resolution path steps. These steps are *not* identical to what is called "eliminations" traditionally, without explaining the *how* and *why* too much at this point. This is a topic for later chapters.

Before details of the BERT resolution steps can be discussed, some more definitions are needed.

Definition 2.9. (CALCULATED) LINK EQUATION
Any set of variables $LE = \{p_1, \ldots, p_n\}$ build a *link equation*, if

1. all variables have values 0 or 1.
2. the condition

$$\sum_{p_i \in LE} p_i \leq 1$$

is verified by an appropriate calculation.

Naturally all subsets of *native base equations* are *link equations*, but the definition is not restricted to that case. Situations where this makes sense are shown later. Note that this and the following definitions refer to "variables" and not just candidates.

Remark 2.10. The term *linked candidates* is also used for a *link equation* regardless how many candidates are involved. The term is used in a generalized manner unlike the well known bi-value relations between candidates.

Definition 2.11. (CALCULATED) BASE EQUATION
Any set of variables $BE = \{p_1, \ldots, p_n\}$ build a *base equation*, if

1. all variables have values 0 or 1.
2. the condition

$$\sum_{p_i \in BE} p_i \geq 1$$

is verified by an appropriate calculation.

Remark 2.12. If the unqualified term *base equation* (BE) or *link equation* is used, it means either a *native base(link) equation* or a *calculated base(link) equation*.

The are situations where calculated base equations are equal to one. In most cases this is not significant, but there are special cases where it matters. They are called *strict base equations* and all their subsets are *link equations*.

Definition 2.13. BASE SET and LINK SET
For each *base equation* or *link equation* the *base set* or *link set* is the set containing the variables of that equation.
The same descriptors are used for corresponding equations and set.

Now we have three levels of meaning for the term *cell*. First a cell is a placeholder in the Sudoku grid, second a set of candidates and third a cell is a base equation. Although the type of each semantic level is clearly different

from the others, there exits a close correlation. This justifies the use of the same descriptors. The type is determined from the context or in doubt is given explicitly. The same is similarly true for rows, columns and boxes.

Remark 2.14. Equations are freely used as arithmetic expressions, constraints or as sets of candidates. If we speak of an *intersection* of two base equations it means the intersection of the two corresponding base sets, likewise the *union* of equations means the union of the candidate sets. Whether equations are used arithmetically or as sets is context dependent.

Definition 2.15. GROUP VARIABLES
Any set of variables can form a *group variable* and the value of this group is the maximum of all variable values.

The immediate consequences are:

The value of a *group variable* is zero if all member variables are zero and otherwise one.

A group variable is linked to another variable, if all member variables are linked to this variable.

A group variable is linked to another group variable, if all members of the first are linked to all members of the second.

Equations with group variables are weaker equations as the original one. For example an equation

$$d_1 + d_2 + \ldots + g_1 + g_2 + g_3 + \ldots + d_n = 1$$

has less solutions than

$$d_1 + d_2 + \ldots + MAX(g_1 + g_2 + g_3) + \ldots + d_n = 1$$

Remark 2.16. *Group variables* can occur in *base equations* and *link equations* and calculations treat them like any other variables. After some members are glued together into a group, the parts become invisible for subsequent calculations.

Traditional Sudoku solving is working with groups all the time, so this definition seems to be common sense. But this overlooks that the definition is not restricted to box intersections. Member variables may also be group variables itself. Why such generalization makes sense and is useful and even sometimes necessary, will be explained where it is relevant.

2.3.4 *In-Equations*

In the following the term *equation* is used intentionally even if it is actually an in-equation of the type ≤ 1 or ≥ 1. These in-equations can always be padded by a special interval variable to form an actual equation, because the missing part is explicit for all cases. This is true because in the BERT context all in-equations are derived in possibly multiple steps from *native base equations*.

$$d_1 + \ldots + d_9 = 1 \;\Rightarrow\; d_1 + d_2 + d_3 \leq 1 \quad \Leftrightarrow \quad d_1 + d_2 + d_3 + [0,1] = 1$$

All additions and subtractions performed during pattern calculations can be done using appropriate interval values and yielding the same result. But the increased effort does not bring more benefit. So the use of the greater/less symbols should be seen as shorthand for the full equations.

2.3.5 *Naming Conventions*

BERT uses the short nrc-notation for candidates: 123 is the candidate for number 1 in row 2 and column 3. Grouped candidates are enclosed in

round brackets: e.g. $12(789)$. Alternatively groups can be written as sum. e.g. $12(789) = (127 + 128 + 129) = (12(79) + 128)$.So terms without letters always denote candidates or groups of candidates.

Base equations, *link equations*, *base sets* and *link sets* have the same names. The type is determined by the context or explicitly mentioned.

Cell at row r and column c : *R*rCc (e.g. *R*1*C*2 is cell in row 1 and column 2)

Row r of number n : *nRr* (e.g. 7*R*5 is number 7 of row 5)

Column c of number n : *nCc*

Box b of number n : *nBb* (Boxes are indexed from top left to bottom right)

Several related objects can be written in an abbreviated manner for convenience. The term *R*1*C*389 denotes the three cells *R*1*C*3, *R*1*C*8, *R*1*C*9, the term 127*R*8 is equal to 1*R*8, 2*R*8, 7*R*8.

This is part of the full BERT notation defined in section 4.1 .

2.4 PATTERNS AND ELIMINATIONS

KEY POINT *What is a meaningful elimination ?*

Traditional Sudoku solving works with a set of *methods*, sometimes also called *rules* or *patterns* synonymously. So far we avoided conflicts with traditional terminology, but the term *pattern* will be defined in a different way in the BERT context. Most methods reflect frequently occurring situations. Traditional methods can be described roughly like this:

A method is a logic fragment describing conditions for some related constraints. If these conditions are satisfied some associated candidates can be cleared. The proof that such a method is working correctly, is done only once using the defining conditions. So a solver only needs to check, if the entry conditions for a method are satisfied in the current situation. There are simple methods having a fixed number of equations and conditions, such are *pairs*, *triples*, *XYZ-wing*s. Other have a dynamic size, like *alternate implications chains*, *nice loops* or *braids*. Many methods inventions are motivated to support manual solving, but especially the dynamic methods can be well beyond the capability of humans.

A lot of work was done trying to name and classify methods, but there seem to been no consistent system that brings methods completely into a meaningful order. A different systematic approach was done by DB developing nrczt-chains, whips, braids with many extensions. Unfortunately the braid system does not solve all known Sudoku, not even with some extensions.

The term *pattern* in BERT will be used for individual arrangements of candidates that satisfy some conditions. These may or may not be associated with traditional methods. To explain this we need an excursion into *sub-puzzles* that will replace methods somehow.

2.4.1 *Sub-Puzzles*

Take a Sudoku with a unique solution. Consider any resolution state where are still unresolved candidates of number x. There is a rarely used traditional method called *pattern overlay*[3], where all valid configurations of number x are stacked on top of each other. Candidates that match none of these configurations can be cleared. The interpretation preferred here views such method as a sub-puzzle consisting of all base equations of number x. Each valid configuration of this sub-puzzle is a solution of the defining base equations. We can extend this idea to sub-puzzles of two, three or more numbers, but there is a little more to consider. The base equations of numbers

[3] see http://sudopedia.enjoysudoku.com/Pattern_Overlay_Method.html

$x, y, z, \ldots$ need to completed by all link equations containing the numbers. Again we build all solutions of this bundle of equations.and probably find many of them, because with less constraints than the original puzzle there is no uniqueness any more.

Now we investigate these solutions for common properties. If for example a dedicated candidate is zero in all of the solutions, this candidate must also be zero in the unique solution of the whole Sudoku. Why? If we add the constraints we ignored for a while by and by again, the number of solutions will shrink and finally arrive at the unique solution. None of the missing constraints can create a contradiction, because we know that one solution exists definitely. What we do not know whether we can find a method in our method repository that can justify the elimination of the candidate. We may consider that "finding all solutions of number plains $x, y, z, \ldots$" is a method itself, but this is very questionable. The problem is not the correctness but the redundancy of the construction.

If on the other hand a dedicated candidate is 1 in at least one of the solutions of the sub-puzzle and zero in the full puzzle, it is absolutely impossible to find an elimination method for that candidate using only numbers $x, y, z, \ldots$. This is because the selected solution would be always a counter-example for any attempt to apply such a method.

Generalizing this idea leads to the definition of sub-puzzles of Sudoku resolution states:

Definition 2.17. SUB-PUZZLE
Let P be the set of equations that describe the relations of all candidates of a resolution state.
S is a *sub-puzzle* of P and likewise P is a *super-puzzle* of S, if S is defined by a set of equations that are also valid in P.

In contrast to a resolution state containing only native base equations, a sub-puzzle may incorporate other types of equations also. Sub-puzzles may contain sub-sub-puzzles. These are also sub-puzzles of the resolution state. The definition of group variables fits in consistently.

Theorem 2.18. *Solution Inheritance*
If an arbitrary expression in a sub-puzzle S is valid for all solutions of S, then this expression is also valid in all solutions of all super-puzzles P.

Proof. Almost trivial. Any solution of P is also a solution of S, because all equations of S are valid in P. So a solution in P that violates the selected expression creates a contradiction. □

Remark 2.19. The *solution inheritance* does *not* imply that a solution of the super-puzzle exists at all. But if a sub-puzzle has no solutions, none of the super-puzzles has solutions.

The idea is to describe eliminations by sub-puzzles instead of methods. A sub-puzzle where a specific candidate is zero in all solutions clearly justifies the elimination of the candidate.

2.4.2 *Equation Minimality*

Any elimination candidate of a resolution state can be confirmed by constructing a sub-puzzle by accumulating more and more equations. This is absolutely trivial but also useless.

Sub-puzzles related to the logical network of many traditional methods cannot be reduced by any candidate. If we build a sub-sub-puzzle with one candidate less, the target(s) can reach non-zero values. We already presented a previous attempt for an abstract elimination pattern definition named *universal elimination*[4]. An *universal elimination* is any minimal set of

[4] see Sudoku forum

candidates capable of justifying an elimination of one target. It is an attempt to have a single abstract definition of an elimination pattern. This definition runs into trouble for various reasons. But nevertheless the idea points into the right direction.

Provided that the elimination is specified by a set of base equation and a set of link equations, a minimal candidate elimination has a number of interesting properties.

1. Each candidate is member of at least one base equation or is a target.
2. Each candidate is member of at least one link equation.
3. Each candidate is linked to at least one candidate of a different base equation or to a target.

Candidates that do not comply to these conditions are dispensable, because they have no impact on the elimination target. This constitutes a contradiction to the minimality.

The disadvantages of this approach is first that minimality is not easy to prove in large configurations and second that other kinds of redundancy may be still there. Even if there are no redundant candidates there may be redundant equations. This becomes very obvious if native equations are split up into a base and a link. The next figure illustrates the effect of redundancy.

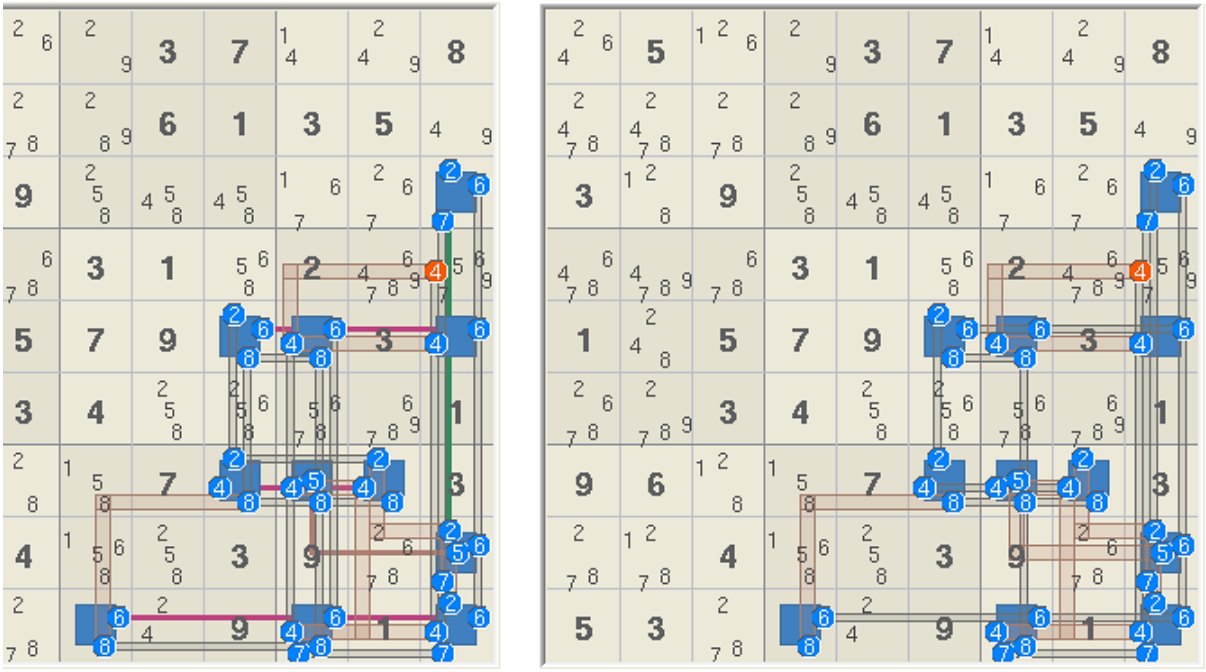


Diagram 1: Equation Redundancy

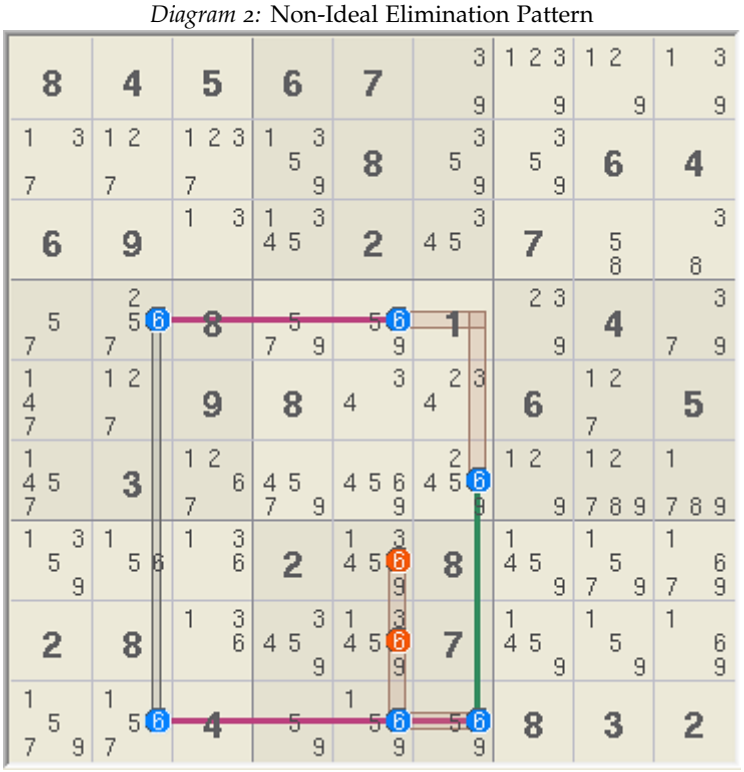
On the left we have all possible equations, 16 native base equations and 17 link equations. None of the candidates is dispensable, so we can only relax some equations. On the right we have a reduced set of equations that also justify $449 = 0$, but only uses 11 base equations and 17 link equations. Now 5 base equations turned into weaker link equations and 5 other link equations are not used at all. All cell equations can be changed to base equations (≥ 1)

and the target is still eliminated. The remaining 11 base equations cover all candidates and are pair-wise disjoint. The resulting minimal sub-puzzle has solutions, where cells contain two true candidates. This is not an error.

Although both diagrams are logically correct, one would prefer the version on the right with removed redundancy. The left diagram is over-defined relative to the target and contains some unnecessary parts. Such effect occurs only in large patterns and is probably due to the general over-defined property mentioned in remark 2.7.

The next example shows a pattern with a property we don't like to have for another reason.

Diagram 2: Non-Ideal Elimination Pattern



The pattern of diagram 2 consists of 3 base equations and 3 link equations. The configuration has minimal candidates, but there is still something odd. Besides the group of targets marked red there is also $692 = 0$ for all solutions of the sub-puzzle. A little investigation reveals that the candidate 692 is eliminable by a sub-sub-puzzle already. The pattern is a concatenation of a *two-string-kite* with $6R4, 6C6$ followed by a *box-line-interaction* with $6R9$.

It seems difficult to make an abstract elimination definition that avoids patterns concatenation of some otherwise independent elimination patterns. In fact *any* eliminable candidate of a Sudoku has eliminations patterns of such kind. We only have to pile up enough equations.

2.5 PATTERN

In both examples the minimality of the reduced set of equations it is still difficult to prove. So additional conditions for patterns would be needed to exclude unwanted effects. This leads to even more difficulties to prove a pattern definition. Because of this problem, that does not go away, we change course.

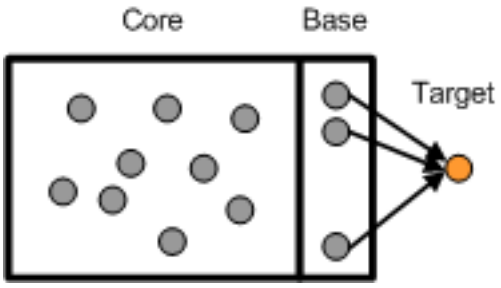
At this point it's good practice to backtrack and recheck all explicit or hidden assumptions. The assumption that turns out to be the most questionable is that a resolution path is a sequence of elimination patterns. This had been the motivation for a general abstract elimination definition. Also the role of

eliminations is overvalued. Of course eliminations are necessary, but they are only intermediate results of an resolution path. BERT introduces other types of intermediate results and their usage.

2.5.1 *Elimination Core*

KEY POINT *An important property common to all eliminations*

Any elimination pattern, even non-minimal ones, can be sub-divided into three sub-puzzles as shown in this schematic picture:



The base part consists of all candidates that are directly connected to the target and the equation $base \geq 1$ is true for all solutions. This is complementary to $t = 0$ for all solutions. So every elimination pattern is associated with a base equation (not native). The logical network of remaining candidates build the *core*. Now we can split the elimination sub-puzzle into two smaller ones. The sub-puzzle "core + base" assures $base \geq 1$ even after the target has been eliminated. The sub-puzzle "base + target" alone is enough for $t = 0$. Both are sub-sub-puzzles of the Sudoku itself, so the equations remain valid for any global solution. This is even true if the targets are cleared or candidates of core or base will have a fixed value.

This separation will be important for BERT. The *base* complies to definition 2.11 of *calculated base equations*. Usually "core + base" patterns that have no common target links are regarded as useless, but this is wasting the opportunity to reuse such equations. Later chapters will show how such re-use is managed.

2.5.2 *Generic Pattern*

Instead of an single abstract pattern definition the following pragmatic definition is used as an intermediate construction. The listed types of pattern pick up essential properties of the above, but are at the same time more specialized. There is no attempt to control the redundancy of generic patterns. This is done later on a global level. Generic patterns just "work" and are worth to be looked at.

Definition 2.20. PATTERN
Let S be a *sub-puzzle* of a resolution state defined by a set of base equations and a set of link equations conforming to (1),(2) and (3) of section 2.4.2.
S is an *elimination pattern*, if for the target variables $\{t_1, \ldots, t_n\}$ the expression $t_1 + \ldots, +t_n = 0$ is valid for all solutions of S.
S is a *base pattern*, if for a sub-set of variables $\{b_1, \ldots, b_n\}$ of S the expression $b_1 + \ldots, +b_n \geq 1$ is valid for all solutions of S.
S is a *link pattern*, if for a sub-set of variables $\{l_1, \ldots, l_n\}$ of S the expression $l_1 + \ldots, +l_n \leq 1$ is valid for all solutions of S.

Some of the generic patterns will be refined to BERT patterns in the next chapter. This will be achieved by some construction principles that allow easy verification and thus avoiding the problems discussed above. It will turn out that very few fundamental pattern types and their combinations will be sufficient. The scope of eliminations with such patterns is still very large. In

a way BERT is inspired by a very special interpretation of the requirements section 2.2.1.

3 PATTERN CALCULATION

This chapter comes with a number of examples demonstrating various types of calculations for base equations and eliminations. The calculations will use only arithmetic additions and subtraction together with reductions stemming from the candidate value restrictions. These examples are generalized to build a formal definition of BERT.

3.1 EASY EXAMPLES

3.1.1 *Pairs, X-Wings*

If four candidates a_1, a_2, b_1, b_2 satisfy the two native base equations $a_1 + a_2 = 1$ and $b_1 + b_2 = 1$ and also the two link equations $a_1 + b_1 \leq 1$ and $a_2 + b_2 \leq 1$, the candidates in the link complements t_1, t_2 of both links have the value zero.

We prove this by the following trivial calculation:

$$\begin{aligned} a_1 + b_1 + t_1 + a_2 + b_2 + t_2 &= 2 \\ a_1 + a_2 + b_1 + b_2 &= 2 \\ &\Downarrow \text{ Subtraction} \\ t_1 + t_2 &= 0 \end{aligned}$$

The only remarkable part is the level of abstraction. There is no need to mention cells, rows, columns or boxes explicitly, because the native base equations are identical for all these types. This means that *naked pair*, *hidden pair* and *x-wing* are defined by identical equations, only the variable names are different. We can get this powerful abstraction for many re-formulated traditional methods. This is similar to *super-symmetry* in the DB concept. All patterns that comply with the *pair* equations build a class – the pair class. This does not mean that a *hidden pair* is the same thing as an *x-wing*, but both share properties that are sufficient to justify their eliminations with structural identical equations.

3.1.2 *WXYZ-Wings*

You may find some explanation of this method in *Sudopedia* or elsewhere.

$z_a \ldots$	$z_b \ldots$	$w_1x_1y_1z_1$	$\ldots$	$x_2y_2z_2$	$\ldots$	$x_3y_3z_3$	$\ldots$	$\ldots$
$\ldots$	w_4z_4	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$

This table shows the Sudopedia definition of "Type 1 WXYZ-Wing" in a schematic manner. The candidates w_i, x_i, y_i, z_i are of any different number and "..." denotes any other candidates. Rows 4 until 9 are omitted.

Re-formulating this method with equations we get:

$$\left.\begin{aligned} w_1 + x_1 + y_1 + z_1 &= 1 \\ x_2 + y_2 + z_2 &= 1 \\ x_3 + y_3 + z_3 &= 1 \\ w_4 + z_4 &= 1 \end{aligned}\right\} = 4$$

$$\left.\begin{array}{rcl} w_1 + w_4 & \leq & 1 \\ x_1 + x_2 + x_3 & \leq & 1 \\ y_1 + y_2 + y_3 & \leq & 1 \end{array}\right\} \leq 3$$

$$\Downarrow \text{ Subtraction}$$

$$z_1 + z_2 + z_3 + z_4 \geq 1$$

We can interpret these equations the following way: The capacity of the set of candidates $\{w_i, x_i, y_i, z_i\}$ is 4 true candidates. The capacity of the set $\{w_i, x_i, y_i\}$ in the second section is at most 3 true candidates, so the set $\{z_i\}$contains at least one true candidate and therefore is a *calculated base equation*. We continue calculating:

$$\left.\begin{array}{rcl} z_a + z_b + z_1 + z_2 + z_3 & \leq & 1 \\ z_a + z_b + z_4 & \leq & 1 \end{array}\right\} \leq 2$$

$$z_1 + z_2 + z_3 + z_4 \geq 1$$

$$\Downarrow \text{ Subtraction}$$

$$2 * (z_a + z_b) \leq 1$$

$$\Downarrow \text{ Restriction } 0 \leq z_{a,b} \leq 1$$

$$z_a + z_b = 0$$

This means that the candidates $z_a = 0$ and $z_b = 0$ are eliminated.

Now we have a look at the schema of "Type 2 WXYZ-Wing":

$z_a \ldots$	$z_b \ldots$	$w_1x_1y_1z_1$	$\ldots$	$\ldots$	$\ldots$	w_4z_4	$\ldots$	$\ldots$
$x_2y_2z_2$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$\ldots$	$x_3y_3z_3$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$

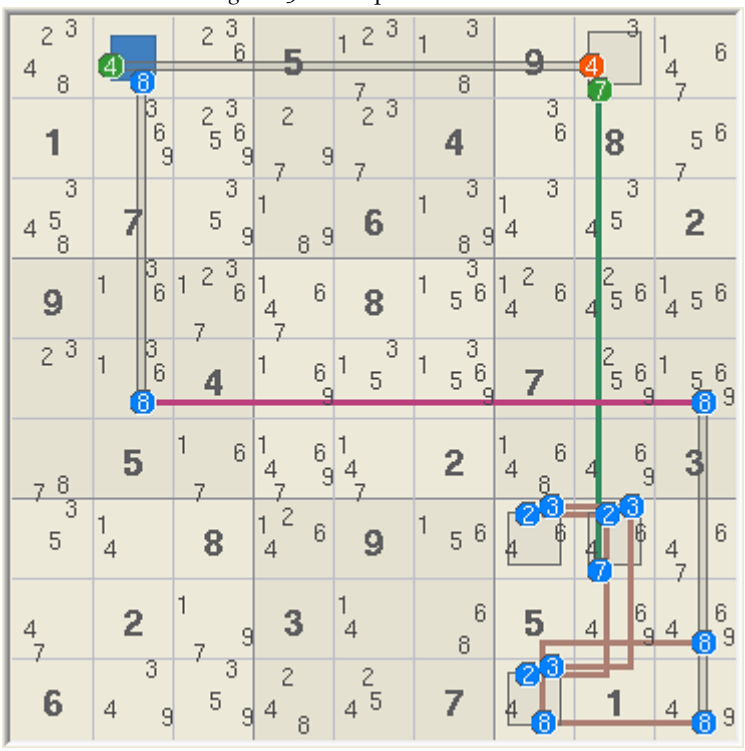
The corresponding equations are exactly the same ones as for type 1. This shows the power of the level of abstraction with equations.

In both variations some of the variables may have zero value without breaking the logic, as long as each equation has at least two variables and at least one target variable is unknown. The kind (row,column,box) of some equations is different, the calculation is not. So we say that conditions of the equation kinds are *decorative* conditions. That does not mean that we can always find sets of variables for arbitrary decorative conditions. This is due to the special connectivity of the Sudoku, but does not degrade the value of the abstraction.

3.1.3 *Free Pattern*

Other traditional methods can be transposed as well, but this is not a main topic of this article. Base equation calculation does not require knowledge of how the pattern was generated or their classification in a method system.

Diagram 3: Example Free Pattern



This pattern can be interpreted as ALS-logic (AB) or as some braid extension (DB) pattern or as anything else you like. The calculation with *base equations* is as follows:

$$\left.\begin{aligned} 2B9 = 277 + 297 + 278 &= 1 \\ 3B9 = 377 + 397 + 378 &= 1 \\ 8B9 = 897 + 8(89)9 &= 1 \\ 8R5 = 859 + 852 &= 1 \\ 7C8 = 778 + 718 &= 1 \\ R1C2 = 812 + 412 &= 1 \end{aligned}\right\} = 6$$

$$\left.\begin{aligned} R7C7 \geq 277 + 377 &\leq 1 \\ R9C7 \geq 297 + 397 + 897 &\leq 1 \\ R7C8 \geq 278 + 378 + 778 &\leq 1 \\ 8C9 = 859 + 8(89)9 &= 1 \\ 8C2 = 852 + 812 &= 1 \end{aligned}\right\} \leq 5$$

$$\Downarrow \quad \text{Subtraction}$$

$$718 + 412 \geq 1$$

final elimination calculation:

$$\left.\begin{aligned} 4R1 \geq 412 + 418 &\leq 1 \\ R1C8 \geq 718 + 418 &\leq 1 \end{aligned}\right\} \leq 2$$

$$718 + 412 \geq 1$$

$$\Downarrow \quad \text{Subtraction}$$

$$2 * 418 \leq 1$$

$$\Downarrow \quad \text{Restriction } 0 \leq 418 \leq 1$$

$$418 = 0$$

With growing size of the patterns drawings with bases and links directly in the Sudoku grid become more and more confusing. An equation diagram gives a more visual but still abstract view on the pattern. Rows are base equations and columns link equations except that in the rightmost column resides the resulting base equation.

	R7C7	*R7C8*	*R9C7*	8C9	8C2	$BE \geq 1$
2*B*9	277	278	297			
3*B*9	377	378	397			
8*B*9			897	8(89)9		
8*R*5				859	852	
*R*1C2					812	412
7C8		778				718

Now we can see clearly that the pattern contain a group link and an ALS. It is also some kind of "queue". But all these properties are not used when performing the above arithmetic calculations. There is no intrinsic ordering of *base equations* in rows or *link equations* in columns. Any sequence of addition or subtraction will yield the same result. Proper ordering can reflect the connectivity of the pattern, but is non-functional for equation calculations. Connections are also related to the grid geometry only if the equations are *native*.

3.2 BASE ELIMINATION

The separation of core and base equation explained in section 2.5.1 leads to a generalization:

Calculation 3.1. BASE ELIMINATION

A *base elimination* consists of

1. one base equations *be*
2. one or more targets t_i
3. each candidate of the base equation is linked to the target(s)

Any elimination calculation can be normalized to a pattern with at most four links of different kinds, $le_{cell}, le_{row}, le_{col}, le_{box}$, corresponding to the four different directions in the Sudoku space. Two links are common, one or three occur sometimes and four are vary rare. The normalization is done by building appropriate candidate groups corresponding to the four directions.

3.3 BASE ARRAY CALCULATION

It is obvious that the core calculation of patterns like diagram 3 can be generalized also.

Calculation 3.2. BASE ARRAY CALCULATION

If

1. n pair-wise disjoint base equations $be_i \geq 1$
2. $n-1$ pair-wise disjoint link equations $le_k \leq 1$

then the *base array calculation* of size n

$$\bigcup_{k=1}^{n-1} le_k \subsetneqq \bigcup_{i=1}^{n} be_i \Rightarrow \sum_{i=1}^{n} be_i - \sum_{k=1}^{n-1} le_k \geq 1$$

provides a *calculated case equation*.

The difference of both equation sets is ≥ 1, that means it is a *calculated base equation*. Note that the calculation does not require native base equations. This little observation seems inconsiderable at first sight, but brings a new idea and finally leads to a radical changed view on Sudoku solutions. Base equations can be calculated from other ones. We will see later that there are even more calculation types to achieve this. So *base equations* play a central role in BERT.

Many of the traditional methods are of the type "base array + base elimination", *Two-String-Kite*, *Wings*, *Death Blossom*, *Queues* of some kinds, etc. The base array calculation is not regarded as method itself, but defines a fairly large class of patterns with a common property.

3.3.1 *Splitting*

Some n-base calculations allow a split-up into smaller parts. This will be also useful in some situations. The pattern diagram 3 divided into a sequence of two calculations, shown as equations diagrams:

	R7C7	*R7C8*	*R9C7*	$BE1 \geq 1$
2*B*9	277	278	297	
3*B*9	377	378	397	
8*B*9			897	8(89)9
7*C*8		778		718

+

	8C9	8C2	$BE \geq 1$
*BE*1	8(89)9		718
8*R*5	859	852	
*R*1*C*2		812	412

This split creates an intermediate base equation $8(89)9 + 718 \geq 1$. There are other possible splits, but the ALS part *R7C7*8 can not be separated.

3.3.2 *Combining*

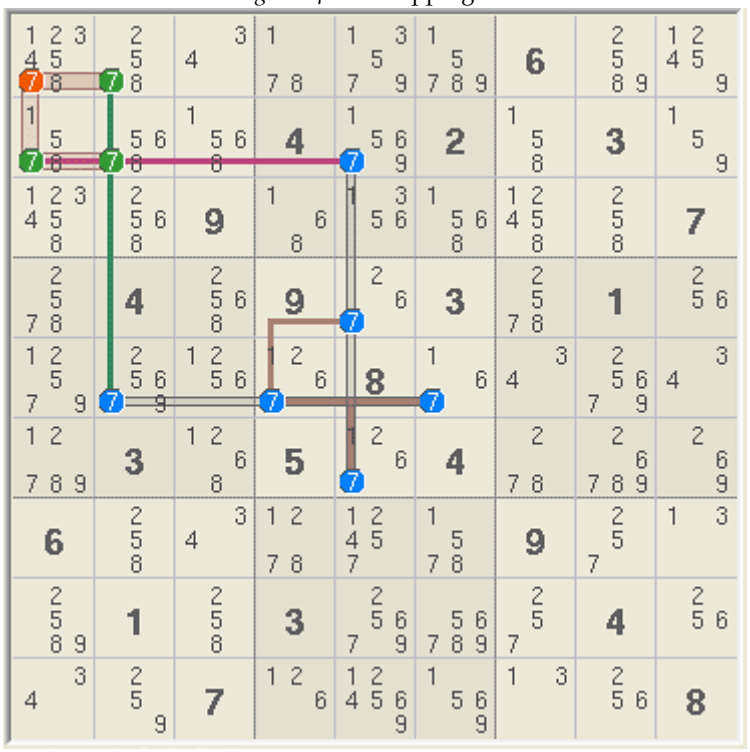
The inverse operation is combining.

If two disjoint base equations $a_1 + a_2 + \ldots + a_n \geq 1$ and $b_1 + b_2 + \ldots + b_m \geq 1$ are linked by $a_1 + b_1 \leq 1$, we get by subtraction another base equation $a_2 + \ldots + a_n + b_2 + \ldots + b_m \geq 1$. This is a direct application of the base array definition.

3.4 BASE REDUCTION

The base array calculation of the previous chapter requires pair-wise disjoint base equations. But the calculation can allow overlapping base equations as long as the overlaps occur in the result only. The next pattern is an example.

Diagram 4: Overlapping Base



The array calculation is as follows:

$$\left.\begin{array}{rcl} 7B5 = 7(46)5 + 75(46) & = & 1 \\ 7R2 = 721 + 722 + 725 & = & 1 \\ 7C2 = 712 + 722 + 752 & = & 1 \end{array}\right\} = 3$$

$$\left.\begin{array}{rcl} 7C5 \geq 7(46)5 + 725 & \leq & 1 \\ 7R5 \geq 75(46) + 752 & \leq & 1 \end{array}\right\} \leq 2$$

$$\Downarrow \text{ Subtraction}$$

$$2 * 722 + 712 + 721 \geq 1$$

$$\Downarrow \text{ Reduction}$$

$$722 + 712 + 721 \geq 1$$

The subtraction result can be reduced to a base equation because all candidate variables are either zero or one.

The general rule for such reductions is:

Calculation 3.3. BASE REDUCTION

Any equation with n integer variables $0 \leq d_i \leq 1$ and integer multipliers $a_i > 1$ is reducible to a base equation.

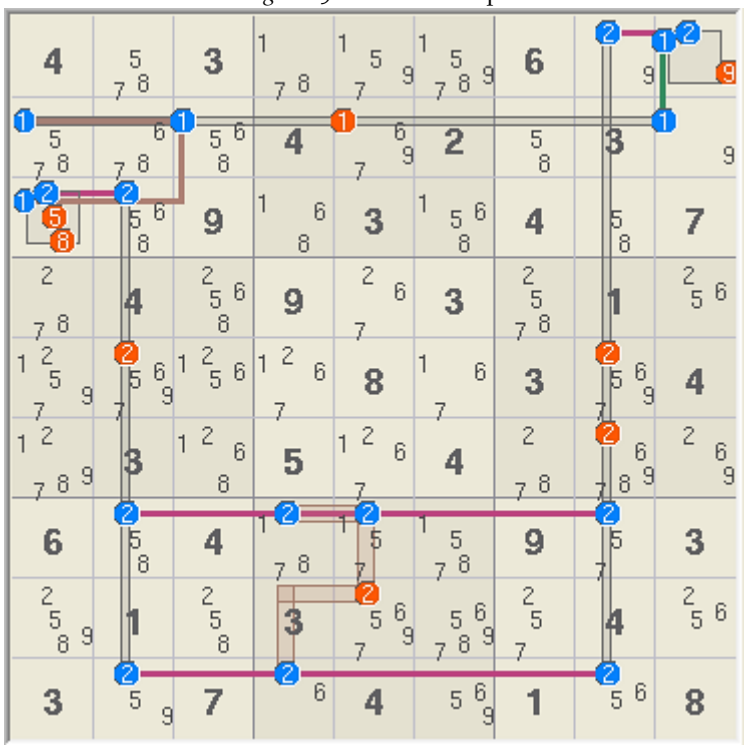
$$\sum_{i=1}^{n} a_i * d_i \geq 1 \Rightarrow \sum_{i=1}^{n} d_i \geq 1$$

So the *base array calculation* has an extended form that is addressed under the same name to keep things simple. The *base reduction* will be used in other calculations too. Most solvers avoid patterns with overlapping base equations to escape complications.

3.5 MATRIX ELIMINATION

Traditional solving knows many patterns that fall under the term *Matrix Elimination* (*Pairs*, *Triples*, *SueDeCoq*, *NiceLoop*, etc). Some call it "base-cover", the AB terminology calls it "rank-zero". The following non-standard example demonstrates the general matrix principle.

Diagram 5: Matrix Example



The corresponding equation diagram with an extra row with elimination candidates:

	2C2	2B8	2C8	R1C9	1R2	R3C1
2R7	272	27(45)	278			
2R9	292	294	298			
2R1			218	219		
1C9				119	129	
1B1					12(13)	131
2R3	232					231
	252	285	2(56)8	919	125	(58)31

Contrary to previous equation diagrams the rightmost column is not labeled with "≥ 1". In fact any of the six columns can play that role (e.g.$R1C9$):

$$\begin{aligned} 2R7 + 2R9 + 2R1 + 1C9 + 1B1 + 2R3 &= 6 \\ 2C2 + 2B8 + 2C8 + 1R2 + R3C1 &\leq 5 \\ &\Downarrow \text{ Subtraction} \\ 219 + 119 &\geq 1 \end{aligned}$$

The following elimination calculation will yield $919 = 0$. So we have six entangled base array calculations.

Performing all six calculations one after another would violate the least-redundancy-principle, because many parts will repeat itself. There is a much simpler combined calculation that does the same thing in one shot.

$$\begin{aligned} 2C2 + 2B8 + 2C8 + 1R2 + R3C1 &= 6 \\ 2R7 + 2R9 + 2R1 + 1C9 + 1B1 + 2R3 &= 6 \\ &\Downarrow \text{ Subtraction} \\ 252 + 285 + 2(56)8 + 919 + 125 + (58)31 &= 0 \end{aligned}$$

The column equations are now used as native(!) base equations too, so they add up to six exactly.

Matrix eliminations relate to the *Dirichlet's*[1] *box principle*[2] or *pigeonhole principle*. It says that applied to this situation: If we have six objects (the true candidates of the six native base equations) and six boxes (the six link equations) then each box must contain one of the objects.

The generalization of matrix calculations:

Calculation 3.4. MATRIX ELIMINATION

A *Matrix Elimination* of size n takes

1. n pair-wise disjoint base equations $be_i \geq 1$
2. n pair-wise disjoint link equations $le_k \leq 1$

then

$$\bigcup_{i=1}^{n} be_i \subsetneqq \bigcup_{k=1}^{n} le_k \Rightarrow \sum_{k=1}^{n} le_k - \sum_{i=1}^{n} be_i = 0$$

It should be noted that the condition (1) – corresponding to matrix rows – does *not* require *native base equations*. The difference equation – corresponding to the targets – is always zero, because all variables must have positive values. Link equations are always $= 1$ in traditional matrix eliminations and eliminate the link complements. The above calculation does not require this and therefore is more general.

3.6 SINGLE ELIMINATION

Although *single eliminations* are the simplest possible elimination patterns and regarded as meaningless, they bring a little surprise when written with equations.

Calculation 3.5. SINGLE ELIMINATION

If there is

1. one variable $s = 1$
2. one link equations $le \leq 1$

then the *single elimination*

$$\{s\} \subsetneqq le \Rightarrow le - s = 0$$

provides a zero equation.

Usually only the situation where the variable is a single true candidate, is regarded as *single elimination*. And no solver will hesitate to eliminate all candidates linked to the true candidate. But the above equation does not define a combined elimination of cell, row, column or box links. Each direction will be treated as separate step and this has some implications

[1] LEJEUNE DIRICHLET 1805-1859 Mathematician
[2] (see http://mathworld.wolfram.com/DirichletsBoxPrinciple.html)

for BERT. The clear reason is that all elimination patterns must be minimal. There should be no exceptions.

The *single elimination* equation also applies to situations, where the variable s is a *group variable*. These are named *Box-Line-Interactions*, *Pointing or Claiming Locked Candidates* traditionally. Here the term *group single* is preferred.

Diagram 6: Group Single

We find $s = MAX(517, 518) = 1$ and $le = 5B3 = 1$ and get as result

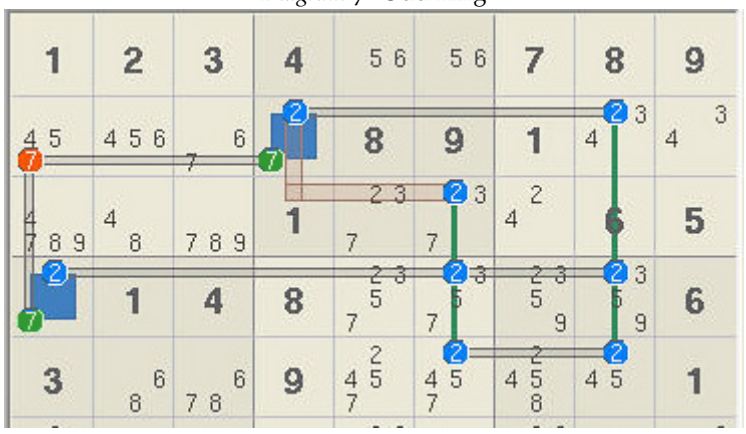
$$le - s = 52(89) + 53(79) = 0$$

From a theoretical point of view *single* eliminations can be interpreted also as *matrix* eliminations of size one.

3.7 ODD RING CALCULATIONS

This chapter picks up ideas from DB and relates to *nrczt-chains*, *whips* and *braids*, but goes beyond these pattern constructions. Many of them are covered by *base array calculations* already, but especially braids may contain odd rings. The next example demonstrates this, comes with a new idea and requires a modified calculation.

Diagram 7: Odd Ring



Without the candidate 236 the pattern could be interpreted as "chain with ALS", but the link $236 + 224 \leq 1$ creates an odd ring. In the DB terminology this pattern is interpreted as *braid[4]*. The assumption of the target to be true leads to a contradiction when performing a series of single eliminations.

The *base array calculation* does not work here, because there are 4 base equations ($R2C4, R4C1, 2C6, 2C8$) and 4 link equations ($2R2, 2B2, 2R4, 2R5$) so their arithmetic difference is useless. The links $7R2, 7C1$ belong to an elimination calculation with a base equation $724 + 741 \geq 1$.

The correct way of calculating this result requires to double the base equation where two links overlap.

$$\left.\begin{array}{rcl} 2 * R2C4 = 2 * 224 + 2 * 724 & = & 2 \\ R4C1 = 241 + 741 & = & 1 \\ 2C6 = 236 + 246 + 256 & = & 1 \\ 2C8 = 228 + 248 + 258 & = & 1 \end{array}\right\} = 5$$

$$\left.\begin{array}{rcl} 2R1 = 224 + 228 & = & 1 \\ 2R4 \geq 241 + 246 + 248 & \leq & 1 \\ 2R5 \geq 256 + 286 & \leq & 1 \\ 2B2 \geq 224 + 236 & \leq & 1 \end{array}\right\} \leq 4$$

$$\Downarrow \text{ Subtraction}$$

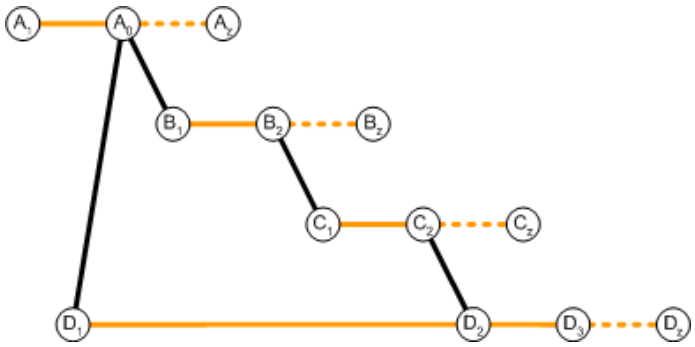
$$2 * 724 + 741 \geq 1$$

$$\Downarrow \text{ Restriction } 0 \leq 724, 741 \leq 1$$

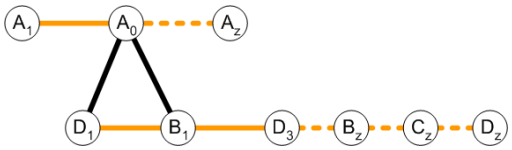
$$724 + 741 \geq 1$$

The *odd ring calculation* is now very similar to the *base array calculation* and subtracts $n - 1$ link equations from n base equations. Using the *base reduction* (see definition 3.3) is always required.

This schematic picture of an odd 7-ring shows more clearly the ring structure. The orange lines denote base equations and the black ones link equations. There is a distinguished base equation at the top that has a variable with two links connected to two other base equations. Then we have alternating link and base equations closing the ring. The top equation will be doubled in the calculation. The optional X_z represent one or more additional variables of each base equation.



With base equations A, B, C, D (pair wise disjoint) we count $2 * A + B + C + D \geq 5$ and diminished by 4 link equations gets $A_1 + D_3 + X_z \geq 1$. Whether or not this does any direct elimination is not important. It creates a new base equation that may be used later on. The impression that the above ring is an ordered structure is not wrong, but only one possible interpretation. The base equation calculation does not use this ordering. The interesting part of the ring structure is the "head" where two links join. The remaining part can be collapsed into a single base equation by performing a *base array calculation*.



Odd rings of any size are transformable into a triangle and produce the same base equation as result. That way we can keep lots of ring variants out of the general definition.

Calculation 3.6. ODD RING CALCULATION
If there is

1. one variable (candidate or group) s
2. one base equations $bs \geq 1$ that contains s
3. one base equation $br \geq 1$ disjoint with bs containing at least 3 variables
4. two link equations $le_1, le_2 \leq 1$ linking s
   and two different variables r_1, r_2 of br

then the *odd ring calculation*

$$2 * bs + br - le_1 - le_2 = 2(bs - s) + (br - r_1 - r_2) \geq 1$$

provides a *calculated base equation*.

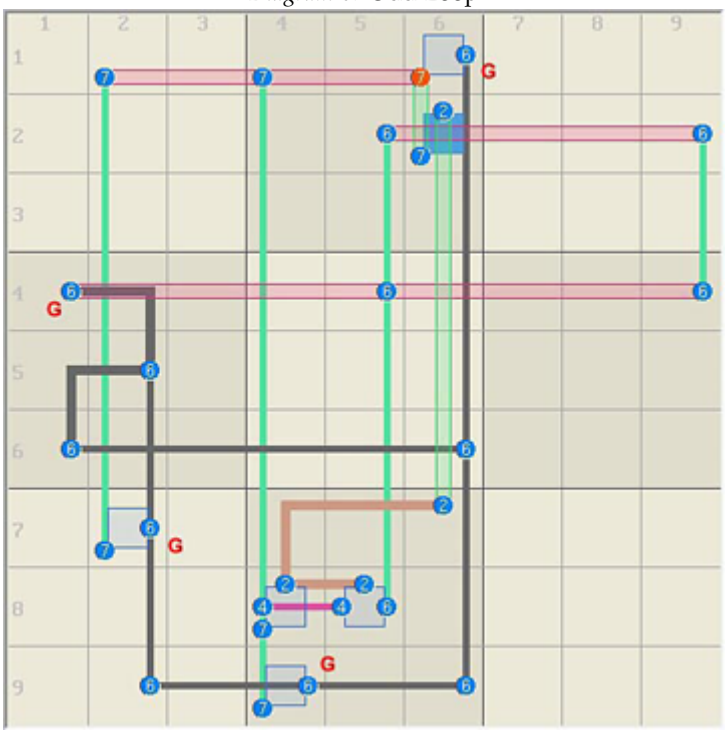
The result is always a base equation after applying a *base reduction* at the end. Odd rings are another important calculation type to produce base equations.

All braid patterns of the DB concept construct odd rings because of the condition that extra links are allowed to "right linking candidates" only. So braids can be evaluated by computing a number of base equations. This is no formal proof but the main idea is to start with the innermost ring, compute a base equation and iterate until no more rings exist.

3.8 ODD LOOP CALCULATION

The example for *odd loop calculation* is taken from an AB web-page[3]. The explanation given there under the name "dark logic" is a bit obscure, but nevertheless the pattern constitutes a valid elimination. Odd loops are very different from odd rings and seem to contain no links.

Diagram 8: Odd Loop



First we focus the attention on the central "dark" part of the sub-puzzle. The key idea for the calculation of *odd loops* is to create link equations *inside(!)* of

[3] http://sudokuone.com/sweb/gen2/blacklog.htm

all base equations where these overlap. This way we get the same number of links and bases.

$$\begin{aligned}
6C2 + 6C6 + 6R6 + 6R9 + 6B4 &= 5 \\
6(46)1 + 652 + 6(579)2 + 66(16) + 69(246) + 6(169)6 &= 5 \\
2 * (652 + 661 + 692 + 696 + 666) \ + \ 616 + 641 + 672 + 694 &= 5
\end{aligned}$$

$$\begin{aligned}
652 + 661 &\leq 1 \\
661 + 666 &\leq 1 \\
666 + 696 &\leq 1 \\
696 + 692 &\leq 1 \\
692 + 652 &\leq 1 \\
&\Downarrow \\
2 * (652 + 661 + 692 + 696 + 666) &\leq 5 \\
&\Downarrow \text{ Odd-Even Reduction} \\
2 * (652 + 661 + 692 + 696 + 666) &\leq 4 \\
&\Downarrow \text{ Subtraction from the bases} \\
616 + 641 + 672 + 694 &\geq 1
\end{aligned}$$

A variable expression with even value that is ≤ 5 is also ≤ 4.

Calculation 3.7. ODD-EVEN REDUCTION

For any equation with $2k + 1$ variables $0 \leq d_i \leq 1$ with

$$2 * \sum_{i=1}^{2k+1} d_i \leq 2k + 1 \Rightarrow \sum_{i=1}^{2k+1} d_i \leq k$$

The result of the final subtraction is a *base equation*. This additionally sheds light on the term *guardian* that is sometimes used traditionally. There are other ways of explaining why at least one of the guardians must be true, but the point here is the integration of *odd loops* into base equation logic.

The remaining parts of the logic is a base array calculation (continuation see section 5.2.5).

Odd loops are fairly rare. This may have reasons like:

- Existing solvers do not search for odd loops.
- Odd loops are really rare.

In any case these patterns do only appear in sufficient hard problems.

3.9 GENERIC ARRAY CALCULATIONS

The *generic calculation* enclose base array and ring calculations as special cases.

Calculation 3.8. GENERIC ARRAY CALCULATION

If there are

1. n base equations $be_i \geq 1$ multiple times b_i
2. m link equation $le_j \leq 1$ multiple times l_j of the same variables

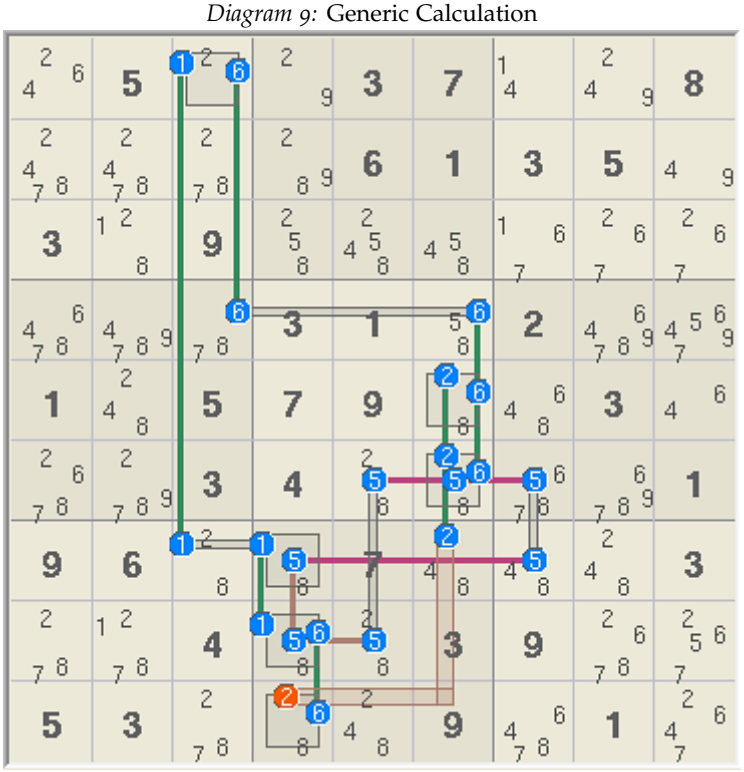
then the *generic calculation*

$$\sum_{j=1}^{m} l_j = \sum_{i=1}^{n} b_i - 1 \Rightarrow \sum_{i=1}^{n} b_i * be_i - \sum_{j=1}^{m} l_j * le_j \geq 1$$

provides a *calculated base equation*.

The result usually needs a following *base reduction* to complete. Only on rare occasions patterns really require a generic calculation. An example that was generated by *Sudoku Explainer* comes here:

Diagram 9: Generic Calculation



The special situation results from two overlapping base equations and two overlapping link equations at the same cell $R7C4$. So the equation 1C4 is used three times, because 574 needs two links to cover both variables.

$$\left.\begin{aligned} 3 * 1C4 = 3 * 174 + 3 * 184 &= 3 \\ 1C3 = 113 + 173 &= 1 \\ 6C3 = 613 + 643 &= 1 \\ 5R6 = 565 + 566 + 567 &= 1 \\ 5R7 = 574 + 577 &= 1 \\ 5B8 = 574 + 584 + 585 &= 1 \\ 2C6 = 256 + 266 + 276 &= 1 \\ 6C6 = 646 + 656 + 666 &= 1 \end{aligned}\right\} = 10$$

$$\left.\begin{array}{rcl} 2 * R7C4 \geq 2 * 174 + 2 * 574 & \leq & 2 \\ 5C5 \geq 565 + 585 & \leq & 1 \\ 5C7 = 567 + 577 & = & 1 \\ R1C3 \geq 113 + 613 & \leq & 1 \\ R5C6 \geq 256 + 656 & \leq & 1 \\ R6C6 \geq 266 + 566 + 666 & \leq & 1 \\ 6R4 \geq 643 + 646 & \leq & 1 \\ 1R7 = 173 + 174 & = & 1 \end{array}\right\} \leq 9$$

$$\Downarrow \text{ Subtraction}$$

$$3 * 184 + 584 + 276 \geq 1$$

$$\Downarrow \text{ Restriction}$$

$$184 + 584 + 276 \geq 1$$

continuing this intermediate base equation result by

$$\left.\begin{array}{rcl} BE = 184 + 584 + 276 & \geq & 1 \\ 6C4 = 684 + 694 & = & 1 \end{array}\right\} \geq 2$$

$$\left.\begin{array}{rcl} R8C4 \geq 184 + 584 + 684 & \leq & 1 \end{array}\right\} \leq 1$$

$$\Downarrow \text{ Subtraction}$$

$$276 + 694 \geq 1$$

The same calculation in BERT notation defined in the next chapter:

$$\begin{array}{rcl} BE = (\ 3 * 1C4, 5R67, 5B8, 16C3, 26C6 \mid & & \\ 2 * R7C4, 5C57, R1C3, R56C6, 6R4, 1R7\ ) & = & (184, 584, 276) \\ [\ (\ BE, 6C4 \mid R8C4\ ) \rightarrow 2B8, R9C4\ ] & = & [294] \end{array}$$

This completes the description of calculation types needed to define BERT. It is important to point out that none of the calculation types introduced in this chapter requires strict base equations $bx = 1$.

4 BASE EQUATION RESOLUTION THEORY

After all this preparations we arrive at the central part. The previous chapter was working with (in)equations of candidate or group variables and arithmetic addition and subtraction operators. There were also reduction operators that were applied to equations. This is level 1 logic and has the power to explain the various types of eliminations of the previous chapters. Now we build level 2 logic on top to describe the resolution path in a condensed manner. This needs a proper notation syntax. I checked existing notations, but none of these fit the purpose required here.

Level 2 logic defines a set of operators that take equations as parameters and have equations as result. The resolution path of BERT will be defined as a number of of level 2 expressions. We use EBNF[1] to formalize the syntax and the semantic is explained when needed.

4.1 NOTATION

The formal description of the notation is a prerequisite to perform computer-supported procedures for scoring (chapter 5) and optimization (chapter 10).

On both levels round brackets are used to designate equations with a value ≥ 1. Square brackets designate equations with value $= 0$. There is no need to designate equations with values ≤ 1, because they occur implicitly only.

4.1.1 *Syntax Backus-Naur*

First some necessary basics.

```
symbol = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
ident = alphabetic_character ,
    { alphabetic_character | symbol | "0" } ;
symlist = symbol , { symbol } ;
```

Level 1 variables:

```
candidate = 3 * symbol ;
candidates = symbol , symbol , "(" , symlist , ")"
           | symbol , "(" , symlist , ")" , symbol
           | "(" , symlist , ")" , symbol , symbol ;
group_elem = candidate | candidates ;
group = candidates
      | "(" , group_elem , { "+" , group_elem } , ")" ;
```

Level 1 constructions:

```
equ_elem = group_elem | group ;
base_equation = "(" , equ_elem , { "+" , equ_elem } , ")";
zero_equation = "[" , equ_elem , { "+" , equ_elem } , "]";
```

[1] http://en.wikipedia.org/wiki/Extended_Backus-Naur_Form

Level 2 variables:

```
cells   = "R" , symlist , "C" , symbol
        | "R" , symbol , "C" , symlist ;
columns = symlist , "C" , symbol
        | symbol  , "C" , symlist ;
rows    = symlist , "R" , symbol
        | symbol  , "R" , symlist ;
boxes   = symlist , "B" , symbol
        | symbol  , "B" , symlist ;
primes = cells | columns | rows | boxes ;
item = primes | ident | expression ;
itemlist = item , { ",", item } ;
baselist = itemlist;
exolink = symlist , "B", symbol , ( "R" | "C" ), symbol ;
linklist = primes , { ",", primes } ;
```

Level 2 constructions:

```
array = "(" , baselist , "|" , linklist , ")" ;
loop = "(" , baselist , "|" , "(*)" , ")" ;
expression = array | loop ;
matrix = "[" , baselist , "|" , linklist , "]" ;
exocet = "[" , baselist , "|" , cells , exolink , "]" ;
elimination = "[" , item , "->" , linklist , "]" ;
assignment = ident , "=" ,
    ( expression | matrix | exocet ) ;
```

4.1.2 *Semantics*

The equations of level 1 are variables of level 2 logic. There are three types. Beside the two well-known types *base* and *link* we have a third type *zero*. A zero equation corresponds to a level 1 equation, where the sum of it's variables is zero.

The members of a `baselist` are always of type *base* and are syntactically separated by a comma. Expressions are usually of type *base* and can only used as *link*, if the value is $= 1$. The type of `assignment` is inherited from the right side always. The items of a `baselist` are always of type base and the items of a `linklist` of type link even if the value is $= 1$.

- Type *zero*: (square brackets) `matrix, elimination`
- Type *base* or *link*: (round brackets) `expression`

The cardinality on an `itemlist` is the number of expanded `prime` elements plus the number of `ident` and `expression` elements. The cardinality of the `linklist` of a valid `array`[2] expression *must* be one less than the cardinality of the `baselist`. This condition is mandatory on each sub-expression of a nested expression. A valid `matrix` or `exocet` requires equal cardinality for base and link and a valid `loop` needs an odd sized list.

4.1.3 *Remarks*

It is important that the notation on level 2 is never using candidates or groups in expressions. This makes expressions somehow independent of particular resolution states. (section 9.4)

In most case there is no choice for the elements of an `array linklist` or of a `matrix linklist`. But despite of that I decided to list the links even in cases

[2] The term *array* denotes the generic form and covers standard array and odd rings as well.

where base equations determine the links completely. Expressions would become complete unreadable for humans otherwise. For pairs, triples and simple chains e.g. there is no choice for the links. Each link must match at least two bases.

4.2 ALGORITHMS

Now we describe how this syntax is connected to level 1 logic.

The procedure `expression` needs extensive explanations. The definition is recursive as expressions may depend on other expressions and is related to *generic calculations* of the previous chapter (section 3.9). We ignore the special cases at the moment.

The algorithm skeleton for expression calculation in some kind of pseudo code:

```
FUNCTION expression ( baselist,linklist )
   candidates = EMPTY LIST
   // base accumulation
   EXPAND baselist TO bases
   FOREACH item IN bases
      IF item IS native
         FOREACH var IN item   // candidate or group
            ADD var TO candidates
      ELSEIF item IS expression OR ident // recursion
         FOREACH var IN expression(item)
            ADD var TO candidates
   // link subtraction
   EXPAND linklist TO links
   FOREACH link IN links
      FOREACH var IN link      // candidate or group
         IF cd EXISTS IN candidates
            REMOVE cd FROM candidates
   // base reduction
   REMOVE DUPLICATES FROM candidates
   RETURN candidates
```

Informally speaking it says that we first accumulate the candidates of all denoted base equations and sub-expression is a list. Candidates may occur multiple times if bases occur multiple times or overlap. Then we reduce the list with candidates denoted by link equations, if and only if they are in the list. Only at the end we remove the remaining duplicate candidates. That assures that we get a *base equation* (≥ 1) finally. Sub-expressions are expanded on demand.

A `matrix` is never re-used in other expressions but may rely on sub-expressions (section 3.5). The cardinality of `baselist` and `linklist` must be equal. All candidates or groups that occur in base equations must also occur in link equations. This assures that we get a *zero equation* [$= 0$] finally. The procedure is also valid for both kinds of single eliminations (section 3.6), where `baselist` and `linklist` contain one item each.

The algorithm for matrix calculation:

```
FUNCTION MATRIX ( baselist,linklist )
   targets = EMPTY LIST
   // link accumulation
   EXPAND linklist TO links
   FOREACH link IN links
      FOREACH var IN link      // candidate or group
         ADD var TO targets
```

```
// base subtraction
EXPAND baselist TO bases
FOREACH item IN bases
   IF item IS native
      FOREACH var IN item   // candidate or group
         REMOVE var FROM targets
   ELSEIF item IS expression OR ident  // recursion
      FOREACH var IN expression(item)
         REMOVE var FROM targets
RETURN targets.
```

An `elimination` is only valid if candidates or groups that occur in link equations are either occuring in the base expression or are part of the common intersection of all link equations (**??**).

```
FUNCTION ELIMINATION ( baseexpr, targetlinks )
   targets = EMPTY LIST;
   FOREACH link IN targetlinks
      FOREACH var IN link      // candidate or group
         ADD var TO targets
   // expr subtraction
   FOREACH var IN baseexpr       // candidate or group
      REMOVE var FROM targets
   REDUCE
   RETURN targets
```

These are the algorithms for the main calculation types.

4.3 RESOLUTION PATH

KEY POINT *The BERT concept relies on principles rather than explicit solving methods.*

4.3.1 *Resolution State*

A *resolution path* is a number of *resolution states* and begins with a start state. A resolution state consists of a set of equations. Scarcely surprising the start state is the set of all 324 *native base equations* together with 17 or more equations of the given candidates and the value restrictions of the candidates of course. Any subsequent resolution state is also a bundle of equations and is possibly extended with more equations. All calculated base equations become persistent part of the succeeding resolution states.

Although it's the human nature to think sequentially and to write sequentially, resolution states are generally not strictly sequentially ordered. They depend on each other, so the proper connection structure is a directed graph reflecting these dependencies.

A resolution state 1 is defined as state after all direct eliminations of the given candidates are executed. This is not mandatory but a convenient practice.

4.3.2 *Resolution Step*

The transition from one *resolution state* to another must obey the principle that the number of solutions is preserved. With BERT the transitions use calculations that produce base equations and zero equations. These calculations are discussed in above. This is not an absolute limitation, but other types of calculations seem to be unnecessary at the time being. A BERT resolution step is a bundle of calculations that determine a zero equation or a base

equation. Zero equations reduce all equations of the current resolution state by stripping off all zero candidates. The final *resolution state* of a Sudoku with an unique solution is reached, when all candidates have a definite value. The only remaining equations are $d = 0$ or $d = 1$ (d any candidate). In case of multiple solutions of the puzzle the remaining equations describe the the sub-puzzle of candidates with no definite values. The BERT concept makes no assumption about solvability or uniqueness of the Sudoku problem. But unique puzzles are clearly the interesting ones.

An explicit arithmetic calculation must exist for any resolution step in BERT. It is not enough that a calculation exists "in principle" or "theoretically". All equations that are assigned to an identifier are part of all subsequent resolution states and can be used with calculations later.

A resolution step is one of the following:

1. a *single elimination* with a candidate or group
2. a *base elimination* with a known base equation
3. a *matrix elimination* with known base equations
4. a calculation of an implicit base equation
5. a calculation of a named *base equation* for later use

4.3.3 *Limits*

The requirement to explicitly list BERT expressions excludes all solution attempts directly based on assumptions. This is the main argument for not using *assumption methods* like *back-doors*[3], etc. A translation to an equivalent BERT expression is impossible.

A number of methods (*Nishio*[4], *Bifurcation*[5], etc) are testing a candidate by assuming $d = 1$ and exploring the consequences. If a contradiction appears after some steps, the candidate can be eliminated (*tertium non datur*). This is also known as *Trial&Error*. These methods create patterns that are usually not minimal (section 2.4.2) and contain redundant parts. Another group of methods is called *coloring*[6]. Here the test does not start at a target but elsewhere and candidates of the pattern are not assigned to definite values but colors.

4.3.4 *Uniqueness*

Uniqueness is a central property of a Sudoku puzzle.

All resolution methods that rely on eliminations can only solve Sudoku that have a unique solution. At the same time these methods verify the uniqueness with a correct resolution path. Only one choice for all cells remains. These methods can also verify the non-existence of a solution, if eliminations lead to a contradiction. But if elimination based methods do not solve a puzzle completely, it's impossible to decide whether the puzzle has multiple solutions or the method set is insufficient.

This calls for an uniqueness check prior to the application of eliminations. This looks perplexing because the check provides the solution already. But the solution is only one among many other properties of the puzzle and from my point of view not very important. The essential properties are related to resolution paths and this is why we should analyze Sudoku puzzles.

[3] see http://sudopedia.enjoysudoku.com/Backdoor.html
[4] see http://sudopedia.enjoysudoku.com/Nishio.html
[5] see http://sudopedia.enjoysudoku.com/Bifurcation.html
[6] see http://sudopedia.enjoysudoku.com/Coloring.html

There is a group of methods that use *uniqueness* of the solution as an assumption. It sparked controversial discussions in the Sudoku community whether this is a good idea nor not. Such discussions are pointless and unproductive without clarifying the purpose of these methods in relation to a resolution theory. There are arguments that Sudoku without a unique solution are not valid Sudoku problems and shift the responsibility to the problem creator. But this is only an exchange of wording and does not alter the fact that verification of uniqueness can only be done through solving.

Uniqueness methods are not useful with BERT. This is not an a-priory decision. At first these methods may give a wrong result for non-unique puzzles, because the assumption of uniqueness cannot prove itself. Second an uniqueness pattern does not build a sub-puzzle where the target(s) are zero in all sub-puzzle solutions. So it is impossible to write the pattern as an arithmetic expression resulting in a zero equation. The uniqueness property is *not* an equation.

5 SCORING

The purpose of *scoring* is to define a metric on calculations, groups of calculations and eliminations and finally on the whole *resolution path*. The term *scoring* is chosen to emphasize the difference to *rating*. The rating value is determined by the most complex elimination step only. There are some variants, the most common is SERATE[1] by GSF. The rating value is intended to reflect the "difficulty of the Sudoku problem" by rating the resistance to the resolution power of the *Sudoku Explainer* solver.

The scoring presented in this chapter is one of other possible ways defining a resolution metric on top of *base equation* calculations. Only properties of the elimination patterns itself are used, never properties of the methods that created the patterns. To keep such metric consistent, a number of principles should establish the scoring rules. The principles chosen here seem rather natural, but nevertheless raises some subtle issues.

The BERT scoring does not evaluate "difficulty" which is always relative to a solver concept, but calculates a *linear arithmetic complexity* regardless how difficult it is to find the eliminations. This might look strange but the separation of resolution steps from the solver opens new possibilities.

Does scoring rival with rating? The answer is yes and no, but more no than yes. The absolute score minimum of all possible resolution paths is existing and has a definite value, but is practically impossible to calculate. Even the distance to the absolute score minimum is unknown. So the main scoring purpose is comparing paths of the same puzzle.

The scoring counts arithmetic additions and subtractions and these are the units of measurement. The attribute *linear* means that any addition or subtraction always has the same score value anywhere in the calculations. Because of this we cannot expect a close correlation with SER rating values.

5.1 SCORING RULES

The score value is defined by very few rules and is calculated with an algorithms taking the expressions of an resolution path as input.

The scoring rules are guided this principles:

1. Count all arithmetic operations with variables equally.
2. Do not count operations on explicit values. (zero, one)
3. Do not count reductions that origin from value restrictions.
4. Re-use of equations are free of cost.
5. Sub-expressions do not alter the score

5.1.1 *Addition Rule*

If two base equations $a_1 + \ldots + a_n \geq 1$ and $b_1 + \ldots + b_n \geq 1$ are added to $a_1 + \ldots + a_n + b_1 + \ldots + b_n \geq 2$ we count one score point. Only the red addition operation is relevant, the other addition operations on the left side are done beforehand and are not counted again. Additions of absolute values on the right side are never counted. Consequently the score of n added base equations amounts to $n - 1$. Candidates may occur multiple times. The same scheme applies to link equations.

[1] download http://gsf.cococlyde.org/download

5.1.2 *Subtraction Rule*

If a link equation $b_1 + \ldots + b_k \leq 1$ is subtracted from n added base equations $d_1 + \ldots + d_n \geq n$ we count one point for each subtracted variable. The right side subtractions are not counted.

There are very good reasons to treat additions and subtractions differently. If we would treat additions by adding candidates one by one also, we just count the number of candidates. This would ignore any pattern structure. Although the subtraction rule score is equal to the number of candidates in most cases, it's not true generally. If we alternatively score one point for each subtracted equation, we end up with counting equations. The leading idea is counting operations and not components of a pattern.

5.1.3 *Link Rule*

The derivation of a link equation from a strict base equation is always free of cost. This is interpreted as some kind of re-use.

5.1.4 *Zero Rule*

Any addition or subtraction of zero terms does not increase the score. There are some situations where the zero term rule applies. One is the follow-up of the subtraction rule, another reduces all affected equations after an elimination of a candidate. So the deleting zero terms from any number of equations is done free of cost. The zero rule is also applied when combining or splitting zero equations.

A zero equation implies all member to be zero. No cost is counted for such split.

$$z_1 + \ldots + z_n = 0 \Longleftrightarrow z_1 = 0 \wedge \ldots \wedge z_n = 0$$

5.1.5 *Group Rule*

The group creation form k candidates $a_1, \ldots, a_k$ to a variable $MAX(a_1, \ldots, a_k)$ scores $k - 1$ points. A subtraction of a group variable is one point. Many patterns can be calculated with or without building groups. The resulting score is intentionally the same if the group is subtracted only once. The group rule must be applied before any other rule in a calculation.

5.1.6 *Reduction Rules*

All reductions that originate from the basic value restriction of candidates $0 \leq d \leq 1$ are free of cost. (3.3) (3.7)

5.2 SCORING EXAMPLES

This section will illustrate the use of the notation and scoring. The score value of traditional patterns may vary by a small margin depending on the actual situation.

5.2.1 *Score of Array*

The notation of the pattern diagram 3 is

$$[\ (238B9, 8R5, 7C8, R1C2 \mid R79C7, R7C8, 8C29)\ :\ R1C8, 4R1\ ] = [418]$$

For better explanation the expression is broken up into two sub-expressions with an assignment.

$$\begin{aligned} BE = (238B9, 8R5, 7C8, R1C2 \mid R79C7, R7C8, 8C29) &= (718 + 412) \\ [\ BE \rightarrow R1C8, 4R1\ ] &= [418] \end{aligned}$$

The result of the first line is in round brackets indicating that the result is a base equation (≥ 1) and the result of the second line in square brackets is zero.

The core pattern BE contains six native base equations and their addition makes 5 points according to the *addition rule*. Then 13 core candidates are subtracted $8(15)2, 8(589)9, (23)77, (237)78, (238)97$. The candidates are not visible in the above equation but are determined during the calculation. So the combined score calculation for BE is 18 points according to the *subtraction rule*. The elimination on the second line adds two link equations scoring one point (*addition rule*) plus two points by subtracting the candidates of BE. So the total score for the pattern is 21 points.

Now we have a look at the notation of the same pattern split up as shown in section 3.3.1.

$$\begin{aligned} BE1 = (\ 238B9, 7C8 \mid R79C7, R7C8\ ) &= (8(89)9 + 718) \\ BE2 = (\ BE1, 8R5, R1C2 \mid 8C29\ ) &= (718 + 412) \end{aligned}$$

The expression $BE1$ needs 3 additions and 8 subtractions, $BE2$ needs 2 additions and 5 subtractions. Together we have 18 points which is the same amount as the above calculation without split. It is also correct to write the expression as

$$(\ (\ 238B9, 7C8 \mid R79C7, R7C8\ ), 8R5, R1C2 \mid 8C29\ ) = (718 + 412)$$

because the result of the inner expression is a base equation of the outer expression.

The calculation rule for array calculations with non-overlapping bases is always $(b - 1) + c$ where b is the number of bases and c the number of candidates of the core pattern. Building of groups does not change the score value.

5.2.2 *Score of Matrix*

Matrix score calculation is relatively easy. The pattern of diagram 5 shows a $6 * 6$ matrix and the BERT notation is

$$\begin{aligned} [\ 2R1379, 1C9, 1B1 \mid 2C28, 2B8, 1R2, R1C9, R3C1\ ] \\ = [252 + 285 + 2(56)8 + 919 + 125 + (58)31] \end{aligned}$$

and scores 5 (additions of extended links) + 16 (subtractions of base variables) = 21 points.

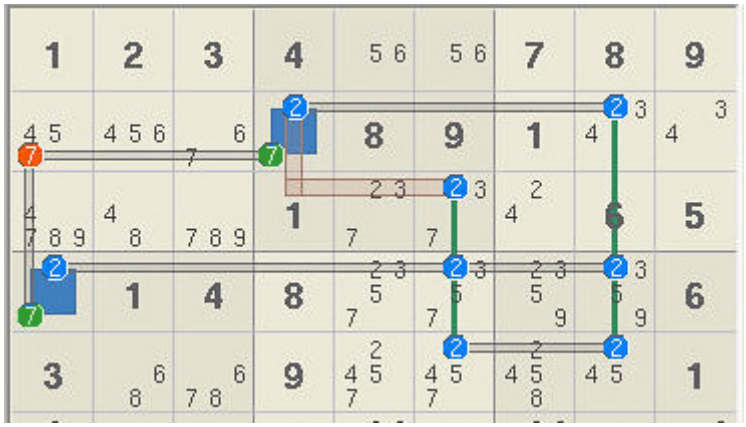
5.2.3 *Score of Single*

Traditionally singles are regarded as totally meaningless, not even worth to be mentioned in a resolution path. The score of a simple *single elimination* is zero points because the compensation of the value one on both sides of the equation is not counted according to principle (2). It is expressly pointed out that there is no need to execute the elimination directions all at the same time. The timing of eliminations is a completely separate issue.

The score of a *group single* (box-line interaction) is equal to the score to build the group. After that the pattern is treated like any other *single*.

5.2.4 *Score of Odd Ring*

The diagram 7 shows an *odd ring*.



We write the notation for the core pattern as

$$\begin{aligned} BE = (\ 2 * R2C4, R4C1, 2C68 \mid 2R145, 2B2) &= (724 + 741) \\ [\ BE \rightarrow 7R2, 7C1\ ] &= [712] \end{aligned}$$

With one doubled base equation we count 4 additions and 9 subtractions. Note that the candidate 224 is subtracted twice! The following reduction is free of cost. With the final elimination of three point the total score is 16 points.

5.2.5 *Score of Odd Loop*

The diagram 8 shows an odd loop. The notation for such patterns have a special syntax:

$$LP = (\ 6C26, 6R69, 6B4 \mid (*)\ ) = (616 + 641 + 672 + 694)$$

The word $(*)$ represents a special link list that is used only with odd loops. It indicates that the link items are the same as the base items and cover the overlapping parts of the base equations. All link candidates of the loop are subtracted twice to cancel the corresponding base equation variables. So we count 4 additions and $2 * 5$ subtractions for the loop equation. The core of the whole pattern is now an array calculation with the loop equation and seven more base equations.

$$\begin{aligned} BE = (\ & LP, 7C24, 6C59, R2C6, 2B8, 4R8 \\ & \mid 6R24, R7C2, R8C45, R9C4, 2C6) = (71(24) + 726 + 616) \\ & [\ BE \rightarrow 7R1, 7C6, R1C6\ ] = [716] \end{aligned}$$

The array calculation takes 7 additions and 18 subtractions. The final elimination calculation starts with building a group scoring one extra and reducing the number of links to the target. Thus we have 2 link additions and 3 subtractions of variables. All together we have a total score of $14 + 25 + 6 = 45$ points. This is among others an example where building a group saves one score point.

The notation of the pattern as an expression with all parts joined together and the same score value:

$$\begin{aligned} [\ (& (6C26, 6R69, 6B4 \mid (*)\ ) \\ & , 7C24, 6C59, R2C6, 2B8, 4R8 \\ & \mid 6R24, R7C2, R8C45, R9C4, 2C6) \\ & \rightarrow 7R1, 7C6, R1C6\ ] = [716] \end{aligned}$$

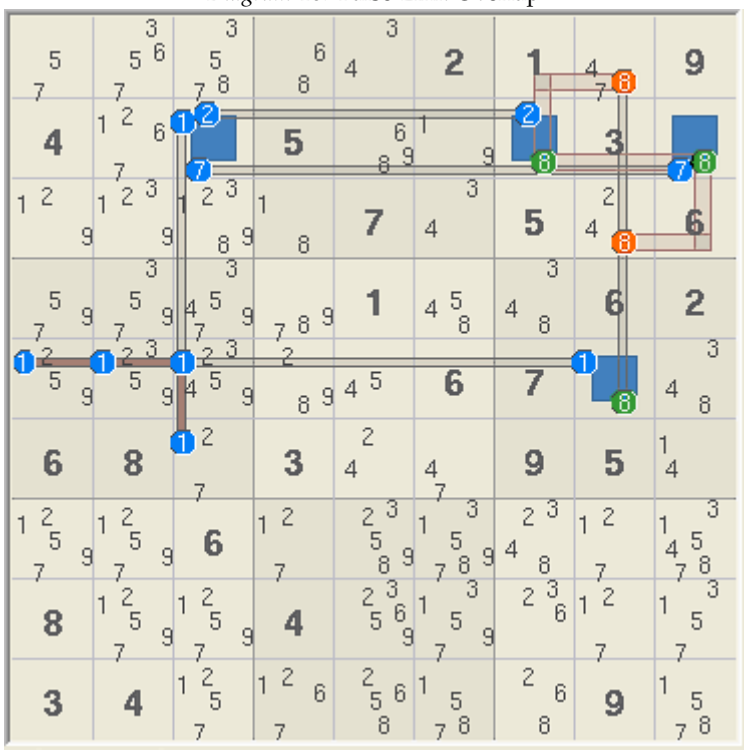
6 EXPRESSION CALCULATIONS

This chapter is meant to discuss more complex patterns that usually only appear in "hard" Sudoku problems. The previous examples are not at all "simple" and way beyond average human solving capability but do not expose the full power of the BERT concept.

6.1 GROUP LINKS

This example shows two different aspects of groups and link equations that do not overlap.

Diagram 10: False Link Overlap



The level 2 notation of this pattern is written as:

$$[\ (\ R2C379\ R5C8\ 1B4 \mid 27R2\ 1C3\ 1R5\ ) \rightarrow 8B3, 8C8\ ] \quad = \quad [8(13)8]$$

An interesting point is the candidate 153, that seemingly is covered by two links. This is not "wrong" logically but leads to nowhere. The calculation procedure uses only variables that still exist at any stage of the calculation. A partial pattern written in level 1 equations demonstrates that the two links do *not* overlap. This is important and may lead to confusion sometimes.

$$\left.\begin{array}{rcl} R2C3 = (127)23 & = & 1 \\ 1B4 = 15(12) + 1(56)2 & = & 1 \\ R5C8 = (18)58 & = & 1 \end{array}\right\} \quad = \quad 3$$

$$\left.\begin{array}{rcl} 1C3 \geq 123 + 1(56)2 & \leq & 1 \\ 1R5 \geq 158 + 15(12) & \leq & 1 \end{array}\right\} \quad \leq \quad 2$$

$$\Downarrow \quad \text{Subtraction}$$

$$(27)23 + 858 \quad \geq \quad 1$$

This is more than a calculation trick. It shows that such patterns require careful investigation, if they are kind of *odd ring* or *base array*. Both may contain overlapped links.

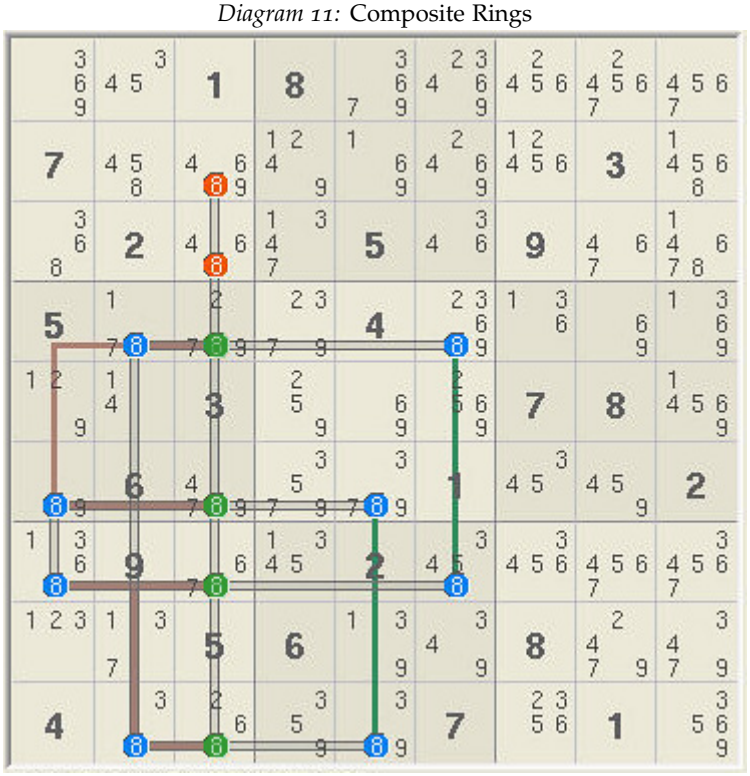
The score calculation of the array part is $(b-1)+c = 14$ points ($b = 5,\ c = 10$). The elimination part scores 4 points, if $82(79)$ is merged into a group.

One might ask why the number of targets appear nowhere in the calculation. This is intentional. Although we have defined score values for patterns only so far, the real purpose is to calculate a score value for a complete resolution path. Any path of a particular puzzle has the same number of elimination candidates in the end. So scoring the elimination candidates would just add a constant value to every path.

6.2 COMPOSITE RINGS

This example shows how calculations depend on each other.

Diagram 11: Composite Rings

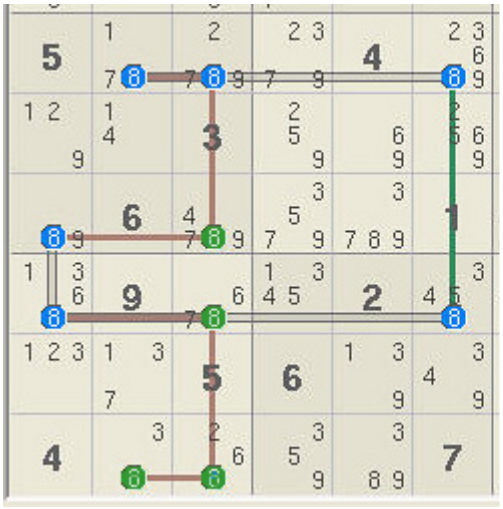


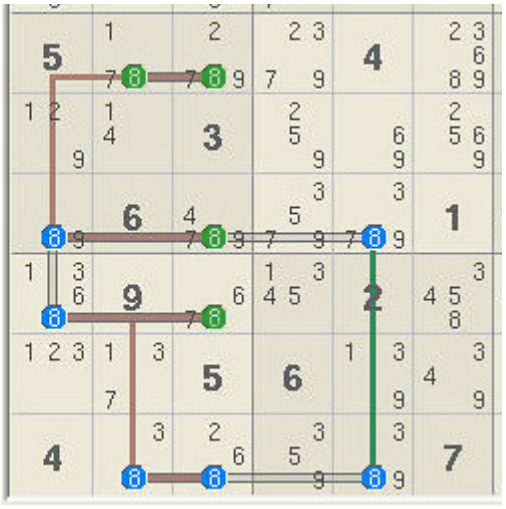
The *xsudo* solver classifies this pattern as *"franken jellyfish"*. Traditional terms are not precisely defined in most cases, but *jellyfish* is somehow related to a *4-matrix*. Unfortunately the matrix calculation relative to the current resolution state is written as

$$[\ 8B47\ 8C56 \mid 8R4679\ ] \quad = \quad [\ ]$$

and results in an empty zero equation. There are no more targets in the four link rows of the jellyfish.

To achieve an explanation for the eliminations $8(23)3$, two partial patterns are helpful.





Both patterns are odd rings and also have some common elements. The notation of the rings is:

$$\begin{aligned} P1 = (\ 2*8B7,8B4,8C6 \mid 8C1,8R47\ ) &= (\ 8(67)3+89(23)\ ) \\ P2 = (\ 2*8B4,8B7,8C5 \mid 8C1,8R69\ ) &= (\ 8(67)3+84(23)\ ) \end{aligned}$$

The *ring calculations* produce (not forgetting the reduction) two base equations. These will be put into the following *array calculation*.

$$[\ (\ P1,P2 \mid 8C2\ ) \rightarrow 8C3\ ] = [\ 8(23)3\ ]$$

To make it more understandable, a translation of the core pattern to level 1 follows. The BERT notation is extremely compact. Note that the *array calculation* requires another reduction at the end.

$$\left.\begin{aligned} P1 = (\ 8(67)3+89(23)\ ) &\geq 1 \\ P2 = (\ 8(67)3+84(23)\ ) &\geq 1 \end{aligned}\right\} \geq 2$$

$$\Downarrow \text{ Addition}$$

$$2*863+2*873+892+893+842+843 \geq 2$$

$$8C2 \geq 842+892 \leq 1$$

$$\Downarrow \text{ Subtraction}$$

$$2*863+2*873+843+893 \geq 1$$

$$\Downarrow \text{ Reduction}$$

$$863+873+843+893 \geq 1$$

The score of both partial rings is $(b-1)+c = 3+7 = 9$. One of the bases is doubled, therefore $b = 3+1$, one candidate is subtracted twice therefore $c = 6+1$. The array calculation scores $1+2 = 3$, $(b = 2,\ c = 2)$. The elimination calculation has only one link. This may happen and is not unusual. With the 4 points elimination score the total score is $2*9+3+4 = 25$.

Of course one can explain this pattern by elementary logic or by generating all solutions as a sub-puzzle or assuming the targets true and find a contradiction. There are four valid permutations of candidate values. Each assigns a different one of the four candidates of the final base equation a true value. But the purpose of this chapter is to show that it is possible to explain the pattern with BERT calculations and make is accessible to scoring.

The notation of the pattern can be written as one composite expression:

$$\begin{aligned} [\ (\ (\ 2*8B7,8B4,8C6 \mid 8C1,8R47\ ) \\ ,(\ 2*8B4,8B7,8C5 \mid 8C1,8R69\ ) \\ \mid 8C2\ ) \rightarrow 8C3\ ] &= [\ 8(23)3\ ] \end{aligned}$$

or even more compact:

$$[\ (\ (8B477, 8C6 \mid 8C1, 8R47), (8B447, 8C5 \mid 8C1, 8R69) \mid 8C2\ ) \rightarrow 8C3\ ] \quad = \quad [\ 8(23)3\ ]$$

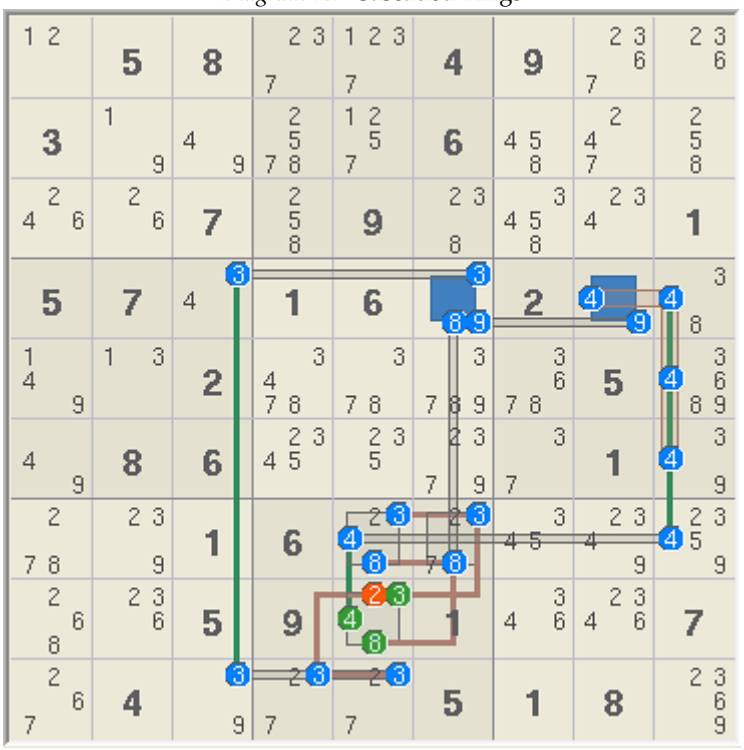
Interestingly the expression does not work without the two sub-expressions. The result is grammatically correct (8 bases, 7 links) but useless.

$$(\ 8B777, 8B444, 8C56 \mid 8C11, 8C2, 8R4679\ ) \quad = \quad (\ 8(49)2 + 8(4679)3\ )$$

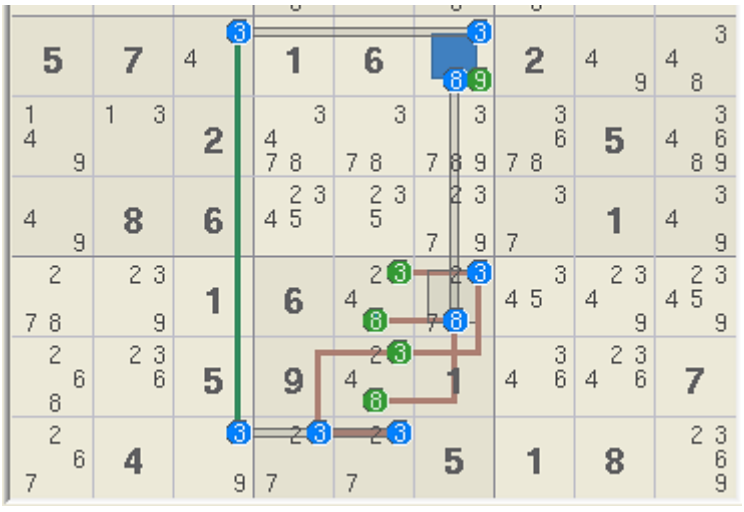
6.3 CASCADED RINGS

This examples shows another type of ring interaction.

Diagram 12: Cascaded Rings



The calculation cannot be done in one step even applying appropriate base equation duplications. We have to start with the innermost ring where two link equations (cell,column) join at 876. Another ring including *BS*1 follows, joining at 475. The next picture shows the isolated inner ring with green colored candidates marking the resulting base.



$$
\begin{aligned}
BS1 = (R4C6, 3C3, 388B8 \mid 8C6, 3R49, R7C6) &= (\ 946 + 3(78)5 + 8(78)5\ ) \\
BS2 = (BS1, R4C8, 4C559 \mid R7C5, 9R4, 4B6, 4R7) &= (\ (348)85\ )
\end{aligned}
$$

The whole pattern in compact notation:

$$
\begin{aligned}
[\ (\ (R4C6, 3C3, 388B8 \mid 8C6, 3R49, R7C6) \\
\quad , R4C8, 4C559 \mid R7C5, 9C4, 4B6, 4R7\ ) \\
\qquad\qquad\qquad\qquad \rightarrow R8C5\ ] &= [\ 285\ ]
\end{aligned}
$$

The score of $BS1$ is $4 + 8 + 1 = 13$ and of $BS2$ is $4 + 10 + 1 = 15$ and 3 points for elimination. This results in 31 total points. Building groups is optional and does not alter the score.

6.4 NETWORK WITH RINGS

This section comes back to diagram 1 as an fairly complex example. The following pattern notation is given without detailed explanations. Decoding such a pattern manually creates considerable headache. So try yourself. An algorithm that does the job seems also to be quite a challenge.

The example is found by the *xsudo* solver, declaring the pattern as *"rank* 5 *logic"*. This only states that the core has 4 extra link equations and is no real help to analyze the pattern.

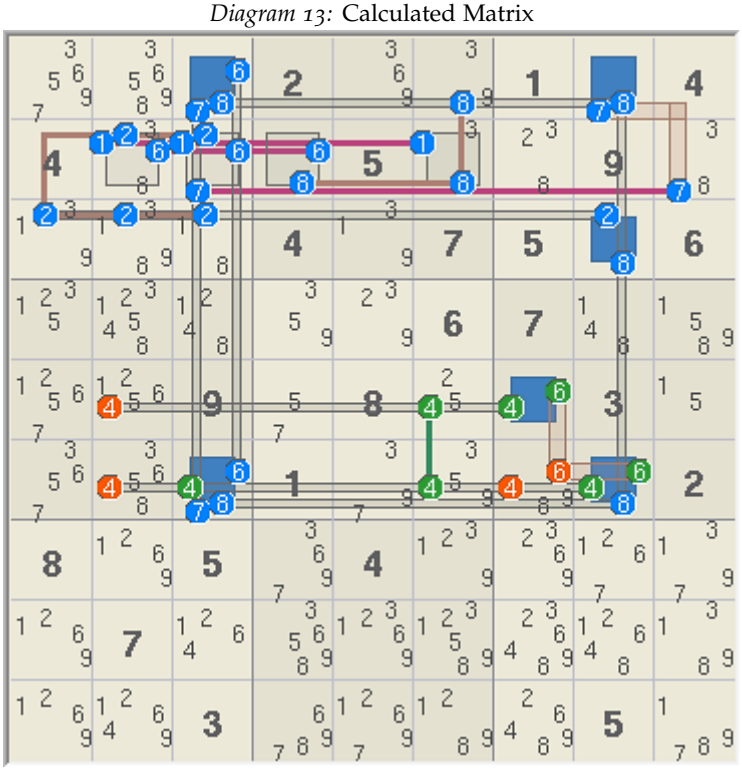
$$
\begin{aligned}
X13A &= (\ 2 * R7C6, R7C8, R9C479 \mid 8B8, 8R7, 2B9, 67R9\ ) \\
X13B &= (\ 2 * R7C7, X13A \mid 4R7, 4C7\ ) \\
X13C &= (\ 2 * R5C7, R5C6, X13B \mid 8R5, 8C7, 2C6\ ) \\
X13D &= (\ 2 * R5C9, R389C9, X13C \mid 276C9, 5B9, 6R5\ ) \\
[\ X13D \rightarrow 4C9, 4B6\ ] &= [\ 449\ ]
\end{aligned}
$$

The pattern score is 54 points.

6.5 CALCULATED MATRIX

Matrix patterns composed from native base equations are well known. The next example shows a 3-matrix with a calculated component.

Diagram 13: Calculated Matrix



The upper part of the pattern conforms to the following BERT expression:

$$MBE = (\ 2*R1C8, 2*R6C8, 2*7R2, R3C8, R16C3, 16R2, 8B2, 2B1 \mid 8C8, 8R16, 67C3, 2*7B3, R2C2346, 2R3\ ) \quad = \quad (463, 468, 668)$$

The pattern contains overlapping links at 818 and 868, so the corresponding bases appear twice. There is no overlap at 623, because 6C3 links only 613 and 663, similarly at 732. So this pattern has some interesting aspects. The resulting base equation forms a *3-matrix* together with 4R6 and R5C7. There are three pairwise disjoint bases and three pairwise disjoint links with the same candidates. Like in any other matrix the links complements are eliminations.

$$[\ MBE, 4C6, R5C7 \mid 6B6, 4R56\ ] \quad = \quad [452, 462, 467, 667]$$

	≤ 1	≤ 1	≤ 1	
	4R5	4R6	6B6	
MBE		46(38)	668	≥ 1
4C6	456	466		$= 1$
R5C7	457		657	$= 1$
	452	46(27)	667	

Remark 6.1. Generally each time there is an elimination pattern with only one target link, the pattern should be investigated for a calculated matrix. The example section 6.2 is not a *calculated matrix*, because the two base equations $P1, P2$ are overlapping. So the matrix $[\ P1, P2 \mid 8C23\ ]$ is not a valid BERT expression. (see definition 3.4)

6.6 EXOCET

Recently the most spectacular discovery of Sudoku solving is the *exocet* pattern. To make *exocet* patterns accessible to scoring, an equation based definition has to be found. This definition may or may not match other

attempts to define such patterns and is not intended to rival with other definitions. It's just the BERT view.

First we split an *exocet* into a central pattern and several support patterns. The central pattern has two base cells sharing a box and either a row or column and two target cells anywhere except in the box and column or row of the base cells. The two base cells usually contain three or four number symbols. *Exocet* patterns with more numbers are theoretically possible but no examples are known.

This schematic diagram shows a configuration with three number symbols represented by a, b, c. The central pattern lives in a box and row (column) combination. The indices X and Y mark the base cell candidates and indices R and S the candidates of the target cells and $abc_i \ldots$ is an abbreviation for $a_i + b_i + c_i[+ \ldots] = 1$. Empty cells are without restrictions. Also some of the candidates may be already cleared.

abc_X	abc_Y	$abc_1 \ldots$	$abc_2 \ldots$	$abc_3 \ldots$	$abc_4 \ldots$	$abc_5 \ldots$	$abc_6 \ldots$	$abc_7 \ldots$
$abc_8 \ldots$	$abc_9 \ldots$	$abc_{10} \ldots$						
$abc_{11} \ldots$	$abc_{12} \ldots$	$abc_{13} \ldots$						

The groups t_R, t_S are the complements of the target cells and will contain the elimination targets. $\boxed{abc_R + t_R}$ $\boxed{abc_S + t_S}$

A pattern is an *exocet*, if the following equations are satisfied

$$\begin{aligned} a_X + b_X + c_X &= 1 \\ a_Y + b_Y + c_Y &= 1 \\ a_R + a_S + \sum a_i &\geq 1 \\ b_R + b_S + \sum b_i &\geq 1 \\ c_R + c_S + \sum c_i &\geq 1 \\ a_R + b_R + c_R + t_R &= 1 \\ a_S + b_S + c_S + t_S &= 1 \end{aligned}$$

The missing $\sum$ index ranges can be any subset of $\{1 \ldots 13\}$ and may be different for each number. All secondary conditions that prevent degenerate configurations are omitted. These have no impact on the calculation.

The three equations in the middle are *base equations* that need to be calculated by some support logic. Although *exocet* patterns are often found with standard support patterns, any base equations satisfying the above conditions will do.

Now we build (weak) groups $A = MAX(a_i)$, $B = MAX(b_i)$, $C = MAX(c_i)$ with the same index ranges. These groups are linked to the base cells ($a_X + a_Y + A \leq 1$ is true) etc. Now we can interpret the *exocet* as special *5-matrix*.

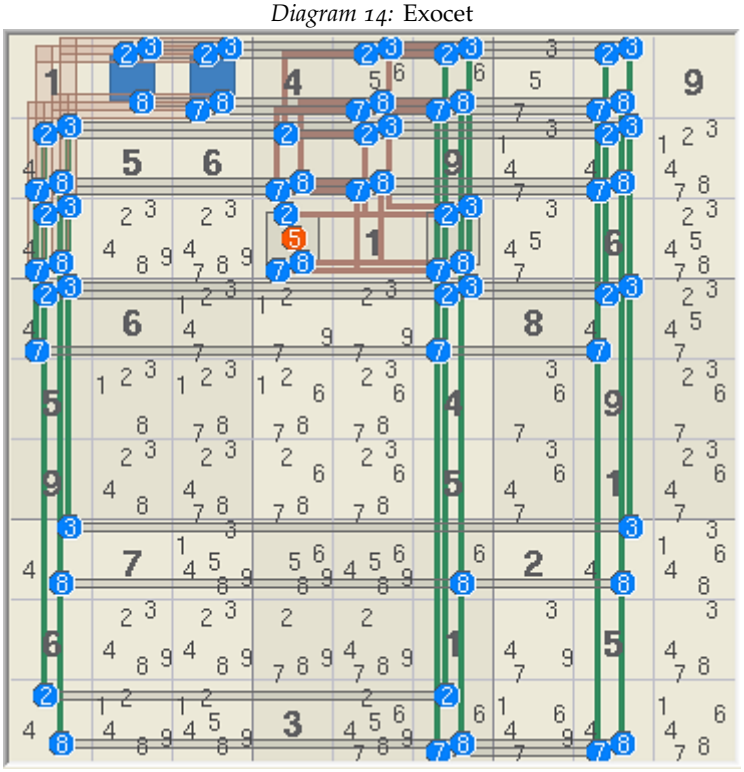
≤ 1	≤ 1	≤ 1	$= 1$	$= 1$	
a_X	b_X	c_X			$= 1$
a_Y	b_Y	c_Y			$= 1$
A			a_R	a_S	≥ 1
	B		b_R	b_S	≥ 1
		C	c_R	c_S	≥ 1
			t_R	t_S	

The sum of all columns of the matrix is ≤ 5, and by subtracting the first five rows we get $t_R + t_S = 0$ taking into account that values are never negative. This complies with the matrix definition (section 3.5). It can be tempting to assume that the first three columns of the matrix also eliminate the remaining row/box candidates (A^*, B^*, C^*). This is a mistake because $A + A^* \not\leq 1$, meaning that A and A^* is *not* linked.

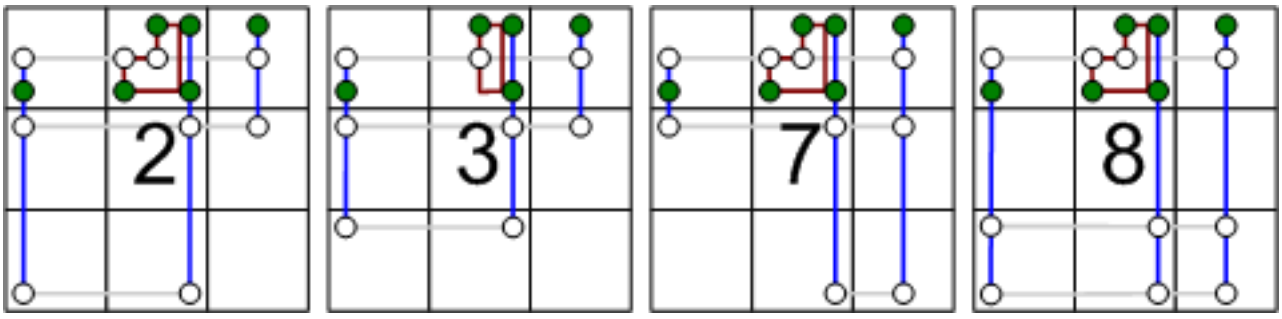
In case the *exocet* pattern has four number symbols, we would get a *6-matrix* analogously.

The abstract *exocet* construction is illustrated with the next example.

Diagram 14: Exocet



The drawing with all bases and links can hardly show the structure of the equations explained above. So we create a separate diagram for each number. The example has four number symbols in the base cells and therefore needs four support patterns each producing a base equation that fits the definition. Consequently we get a *6-matrix* as central pattern.



The resulting base equation candidates are marked green. Each calculation takes three columns and a box as base and three rows as link. The boxes overlap with a column, but the overlapping candidates are part of the result and therefore friendly. We get four support expressions.

$$
\begin{aligned}
S2 &= (\ 2C168, 2B2 \mid 2R249\ ) \\
S3 &= (\ 3C168, 3B2 \mid 3R246\ ) \\
S7 &= (\ 7C168, 7B2 \mid 7R249\ ) \\
S8 &= (\ 8C168, 8B2 \mid 8R279\ )
\end{aligned}
$$

Now we can write the whole pattern as a composite expression. The link section of the matrix uses special syntax elements that occur only in an *exocet* matrix.

$$[\ R1C23$$

$$
\begin{aligned}
&, (\ 2C168, 2B2 \mid 2R249\ ) \\
&, (\ 3C168, 3B2 \mid 3R246\ ) \\
&, (\ 7C168, 7B2 \mid 7R249\ ) \\
&, (\ 8C168, 8B2 \mid 8R279\ ) \\
&\quad \mid R3C46, 2378B2R1\ ] \quad = \quad [\ 534\ ]
\end{aligned}
$$

and calculate the score: $S1: \ (3+9) = 12$, $S2: \ (3+8) = 11$, $S3: \ (3+9) = 12$, $S4 \ : \ \ (3+10) \ = \ 13$. The matrix: $5 + (6+5+6+6+7) \ = \ 35$. The subtractions are in parenthesis. The 7 points at the end come from subtracting the variables from $R1C23$.

The total score is $35 + 12 + 11 + 12 + 13 \ = \ 83$ points showing that *exocet* patterns are pretty expensive.

A number of variants of *exocet* patterns exist, but all of them share the central matrix. The leading base equations are always cells. This is due to the Sudoku construction.

7

EQUATION RE-USE

Re-using intermediate results when solving problems is very natural. The re-use of eliminations in Sudoku solving is pretty common, but usually not even recognized as such. Traditional solving seems to understand eliminations as isolated events that "appear" somehow. From the BERT point of view re-use an elimination in the middle of a resolution path is not an isolated event but the last step of a pyramid of other steps originated from basic equations.

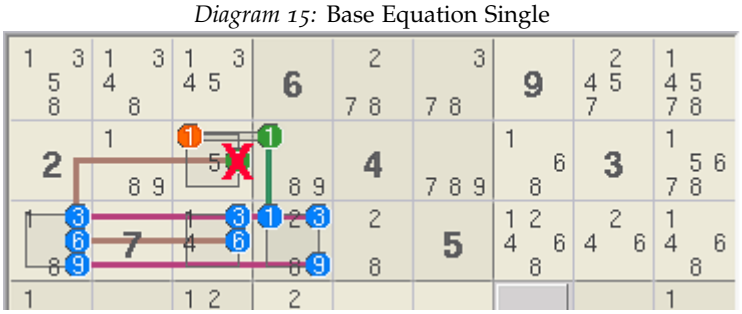
Eliminations are just re-usable value equations of the form $d = 0$. Their only purpose is to justify value equations $d = 1$ or enable further intermediate results. There is no reason to restrict the resolution to elimination re-use only. The BERT resolutions use additionally re-usable equations of the form $a_1 + \ldots + a_n \geq 1$ and $a_1 + \ldots + a_n = 1$. Re-using such equations opens the possibility to find more "efficient" resolution paths. The efficiency is measured by the scoring explained in the previous chapters.

The following examples are manually derived from standard resolution paths. The selection is limited due to lacking solver support. It is likely that there are lots of re-use opportunities in every resolution path.

7.1 RE-USE OF BASE EQUATIONS

Once a base equation is calculated it will indicate a *single elimination* like any native base equation, if all but one candidate of the equations is zero.

Diagram 15: Base Equation Single



The candidate 123 is eliminated by the expression

$$\begin{aligned} A = (\ 39R3, 6B1, 1C4 \mid R3134\ ) &= (124 + 623) \\ [\ A \rightarrow R2C3, 1R2\ ] &= [\ 123\ ] \end{aligned}$$

If it happens that later 623 is cleared, we remember the base equation $124 + 623 \geq 1$ and immediately get $124 = 1 \Rightarrow 134 = 0$. The traditional elimination would be $6(789)3 = 0$ by box-line interaction and then eliminating 134 by a triple. This is not wrong but inefficient by almost any concept of "simplicity".

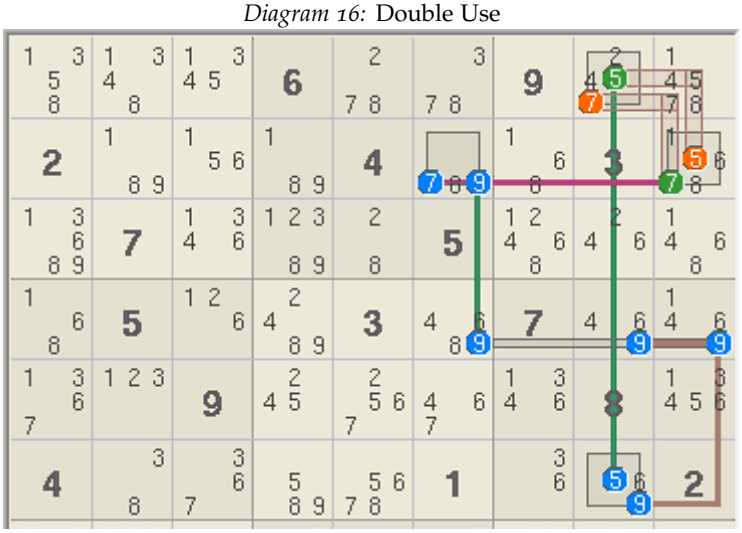
Base equation singles are very valuable in order to minimize the global score.

7.2 DOUBLE USE

The next example had been subject of a discussion whether it contains one or two eliminations. With the BERT concept this questions becomes meaningless. The pattern is interpreted as three calculation steps. The

first is a base equation calculation continued by two different elimination calculations.

Diagram 16: Double Use



The BERT interpretation of this pattern constructs an array calculation and two elimination calculations with different links.

$$
\begin{aligned}
B &= (\ 9C6, 9B6, 5C8, 7R2 \mid R2C6, R6C8, 9R4\ ) \\
[\ B \rightarrow R1C8, 7B3\ ] &= [\ 718\ ] \\
[\ B \rightarrow R2C9, 5B3\ ] &= [\ 529\ ]
\end{aligned}
$$

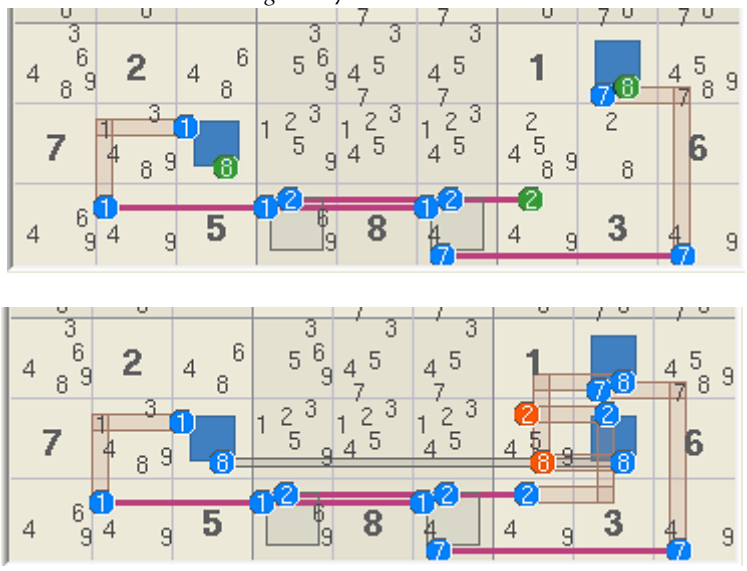
]

The score is 10 points for the base and 3 points for each elimination.

7.3 PARTIAL RE-USE

This example shows how two similar patterns that share common parts are scored.

Diagram 17: Partial Re-Use



The lower diagram shows two eliminations mixed with the common part. Both patterns share the base equation SD with score $(4+9) = 13$

$$
\begin{aligned}
SD &= (\ 127R9, R8C3, R7C8 \mid R9C46, 1B7, 7B9\ ) \\
&= (\ 878 + 883 + 297\ )
\end{aligned}
$$

and are continued

$$\begin{aligned} [\ (\ R8C8, SD \mid 2B9\ ) \to 8R8, 8B9\ ] &= [\ 887\ ] \\ [\ (\ 2 * R8C8, SD \mid 8R8, 8B9\ ) \to 2B9\ ] &= [\ 287\ ] \end{aligned}$$

and the scores are $(1 + 2) + 3 = 6$ and $(2 + 3) + 2 = 7$. The shared part is counted only once of course.

8 PATTERN DECOMPOSITION

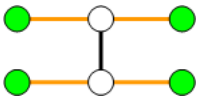
This chapter gives some hints, how to identify base and links in an arbitrary pattern. This information helps translating patterns into BERT notation. More investigation is needed on this topic.

8.1 FRAGMENTS

KEY POINT *What are the atomic parts of a BERT pattern ?*

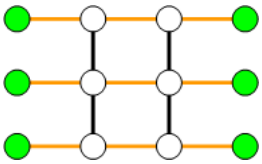
There are always sub-expressions of a pattern that allow no more decomposition. Certain types of such atomic fragments occur frequently. The following pictures show fragments an abstract manner. The green variables are members of the result base equation and contain at least one true variable.

Diagram 18: 2-Base Fragment



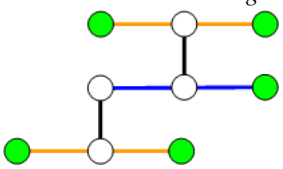
The simplest type is two linked bases. Such fragments are part of chained patterns.

Diagram 19: 3-Base Fragment



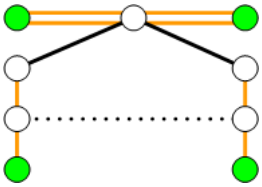
Another fragment type consists of tree bases and two links. Such fragments are found in patterns with ALS. Usually only some of the variables are still existing in a particular resolution state.

Diagram 20: Combined Fragments



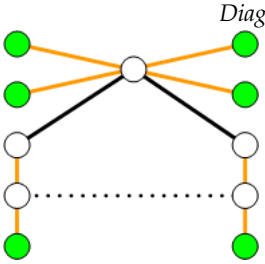
Fragments can be combined if they share bases. The blue base is the shared one. Some of the green result variables always turn to link variables when combining.

Diagram 21: Ring Fragment



A more interesting fragment relates to odd ring patterns. The doubled base is shown at the top. This fragment represents not a ring equation, but combined with other fragments that close the ring along the dotted line, it is the key part.

Diagram 22: Overlap Fragment



Amazingly this fragment for overlapping bases and overlapping links is very similar to the previous one.

It is not intended to give a full list of all possible fragment types. But fragments are very helpful to analyze patterns and for the translation into BERT equations. A fragment contains always all bases that share a common link as a matter of principle. So each link in a pattern related to exactly one fragment whereas the connecting bases appear repeatedly. A complete list of occurring fragments can be derived from any individual pattern.

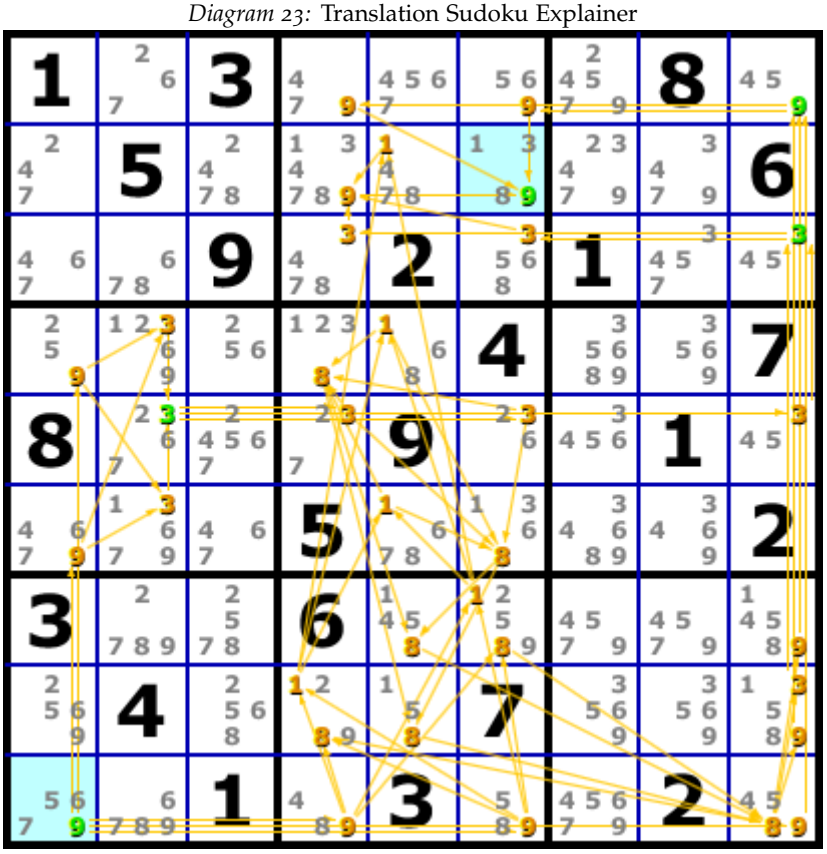
8.2 TRANSLATION

KEY POINT *How to translate arbitrary pattern diagrams into BERT expressions ?*

This example is picked from a list of hardest Sudoku and shows an early elimination pattern generated by the *Sudoku Explainer* program.. The shown pattern does not depend on the previous eliminations.

```
1.3....8..5......6..9.2.1.......4..78...9..1....5....23..6......4...7.....1.3..2.
```

Diagram 23: Translation Sudoku Explainer

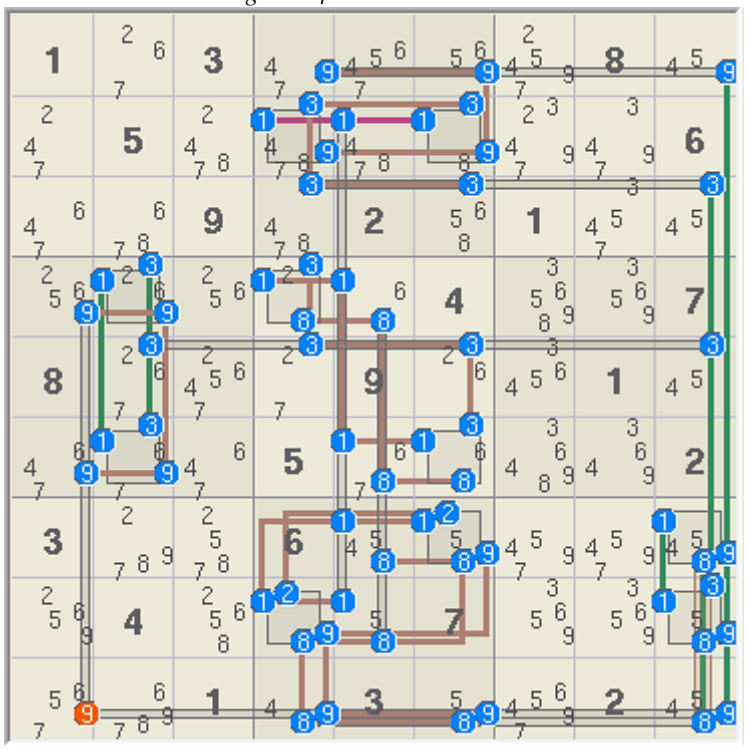


The program labels the pattern as *"dynamic contradiction forcing chain"*. The explanation accompanying the diagram consists of lengthy implication chains that finally construct a contradiction. The conclusion justifies the elimination

of candidate 991. Although logically correct, it's up to ones personal view of a good explanation.

The first problem when trying to translate the pattern into BERT expressions is: how to identify whether arrows in the diagram are bases or links. The explaining text helps a little, because only bases can imply a value 1. Next we discover that some bases or links (mainly cells) not even have arrows, and can only be identified indirectly. But in the end we have a list of bases and links. For verification of the findings we put the result into the *xsudo* program.

Diagram 24: Translation Xsudo



We found 17 non-overlapping native base equations and 16 link equations. Two extra link equations point to the target. The first attempt build a BERT expression is to try a *base array calculation*, because the the number of bases exceeds the links by one. This approach is successful here.

$$\begin{aligned} &(139B2, 139B4, 138B5, 1289B8, 1389C9 \mid \\ &R2C46, R46C2, R48C4, R67C6, R78C9, 18C5, 3R35, 8R9, 9R1) \quad = \\ &(9(46)1 + 99(46) + 999) \end{aligned}$$

$$[(9(46)1 + 99(46) + 999) \rightarrow 9C1, 9R9] \quad = \quad [991]$$

This expression verifies the elimination. At least one candidate of the result has a value 1. It is recalled that the result equation remains valid regardless how many candidates will get definite values later.

As one would expect a rather complex expression for the pattern, the flat array expression is nice and disappointing at the same time. The main intention of the BERT abstraction is not to reflect the connectivity of the pattern or describe all properties, but to identify the preconditions of eliminations and to have a basis for scoring.

8.3 SUB-EXPRESSIONS

The previous section explains that connectivity limits the options of building sub-expressions, if the pattern contains 3-base (or more) fragments. The above example will be taken to demonstrate the limits of sub-expressions.

$$\begin{aligned} BOX2 = (\ 139B2 \mid R2C46\ ) &= (\ 125 + 3(46)3 + 91(46)\ ) \\ BOX4 = (\ 139B4 \mid R46C2\ ) &= (\ 352 + 9(46)1\ ) \\ BOX5 = (\ 138B5 \mid R4C4, R6C6\ ) &= (\ 1(46)5 + 35(46) + 8(46)5\ ) \end{aligned}$$

All three sub-expressions have a 2-matrix with two cell links and there are no sub-expressions possible. $BOX2, BOX4$ and $BOX5$ are classical ALS and 3-base fragments. The three sub-expressions are valid base equations and may be re-used later if appropriate.

A general rule for building sub-expressions is to incorporate all bases that share the same link. So two bases can be combined safely, if they share one link exclusively. Three bases can be combined, if they share two links exclusively. Otherwise the balance of base and link numbers is violated. Exceptions from this rule require overlapping bases and special configurations.

According to the rule $BOX2$ is combined with $39C9$ then with $18C9$ and $BOX5$ with $8B8$.

$$\begin{aligned} BOX2A = (\ BOX2, 39C9 \mid 3R3, 9R1\ ) &= (\ 125 + 3(58)9 + 9(789)9\ ) \\ BOX2B = (\ BOX2A, 18C9 \mid R78C9\ ) &= (\ 125 + 359 + (89)99\ ) \\ BOX5A = (\ BOX5, 8B8 \mid 8C5\ ) &= (\ 1(46)5 + 35(46) + 89(46) + 884 + 876\ ) \end{aligned}$$

Now the sub-expressions are assembled to represent the whole pattern again.

$$\begin{aligned} (\ BOX4, BOX2B, BOX5A, 129B8 \mid 3R5, R8C4, R7C6, 1C5, 8R9\ ) &= \\ (\ 9(46)1 + 99(46) + 999\ ) & \end{aligned}$$

The equivalent equation diagram is more visual and reveals more possible sub-expressions.

	$3R5$	$8R9$	$1C5$	$R8C4$	$R7C6$	≥ 1
$BOX4$	352					9(46)1
$BOX2B$	359	899	125			999
$BOX5A$	35(46)	89(46)	1(46)5	884	876	
$1B8$			1(78)5	184	176	
$2B8$				284	276	
$9B8$				984	976	99(46)

Combining $BOX2B$ and $BOX5A$ gives:

$$BOX2B5A = (\ BOX2B, BOX5A \mid 8R9\ ) = (\ 1(246)5 + 35(469) + 884 + 876 + 999\ )$$

	$3R5$	$1C5$	$R8C4$	$R7C6$	≥ 1
$BOX4$	352				9(46)1
$BOX2B5A$	35(469)	1(246)5	884	876	999
$1B8$		1(78)5	184	176	
$2B8$			284	276	
$9B8$			984	976	99(46)

continued:

$$\begin{aligned} BOX245 = (\ BOX4, BOX2B5A \mid 3R5\ ) &= (\ 1(246)5 + 884 + 876 + 9(46)1 + 999\ ) \\ BOXALL = (\ BOX245, 1B8 \mid 1C5\ ) &= (\ (18)84 + (18)76 + 9(46)1 + 999\ ) \end{aligned}$$

	$R8C4$	$R7C6$	≥ 1
$BOXALL$	$(18)84$	$(18)76$	$9(46)1+999$
$2B8$	284	276	
$9B8$	984	976	$99(46)$

The pattern is now represented by an expression with all sub-expressions collapsed into $BOXALL$.

$$(BOXALL\ 29B8 \mid R8C4\ R7C6) \quad = \quad (9(46)1+99(46)+999)$$

Finally we can write the pattern expression explicitly with all developed sub-expressions. The score is not always affected by building sub-expressions and counts 77 points including the elimination.

$$\begin{aligned} &((((139B4 \mid R46C2) ((((139B2 \mid R2C46) 39C9 \mid \\ &3R3\ 9R1) 18C9 \mid R78C9) ((138B5 \mid R4C4\ R6C6) 8B8 \mid 8C5) \mid \\ &8R9) \mid 3R5) 1B8 \mid 1C5) 29B8 \mid R8C4\ R7C6) \\ &= \\ &(139B2\ 139B4\ 138B5\ 1289B8\ 1389C9 \mid \\ &R2C46\ R46C2\ R48C4\ R67C6\ R78C9\ 18C5\ 3R35\ 8R9\ 9R1) \end{aligned}$$

The resulting expression is hardly human readable but perfectly machine readable. It reflects in some sense the connectivity of the pattern. The nested sub-expressions correspond to nested ALS and chain connections. This example demonstrates the potential of sub-expressions of a fairly complex pattern. Most of them are not needed because there is no re-use later. It's not even sure that the elimination itself has some relevance. Sub-expressions are only useful, if they are a necessary part of the calculation or being re-used in other calculations, but this is not directly obvious in many cases.

9 STRATEGY

KEY POINT *What means "strategy" in a Sudoku resolution theory ?*

We can look at a common definition: *"Strategy is a high level plan to achieve one or more goals under conditions of uncertainty."*[1]

This a good start. One goal is certainly to reach a solution, but without secondary goals Sudoku solving is pointless.

The secondary goals of the BERT concept have three components that reflect the informal requirements stated in section 2.2.1.

1. Only patterns that are candidate minimal are used.
2. Patterns must have a description as arithmetic expressions.
3. Seek the simplest resolution path measured in score points.

There is no real strategy in Sudoku because there is no uncertainty. A Sudoku problem is finite and the resolution path with lowest score exists and can be found theoretically. But Sudoku problems are "hard" and therefore the minimal resolution path can only be obtained by checking all possible cases. This is practically impossible even with excessive computer support. So we have a quasi-uncertainty and a quasi-strategy to approximate the score minimum.

9.1 SIMPLEST FIRST

Manual solving looks for *single eliminations* first, then *box-line-interactions*, then *pairs*, then more sophisticated methods in an almost predefined order. Most solver use a strategy like that, mimic and automate manual solving. This strategy is designed to find the step associated with the lowest "size". Putting aside the lack of a general *size* definition (except for *braids*), most methods or stages of dynamic methods can be arranged according to some partial order. We can call this the traditional strategy.

The BERT scores of resolution paths produced by such traditional strategy are not really bad but not good either. This has a very simple reason.

The leading idea of this strategy is that each elimination brings the solving nearer to the solution, and therefore using steps of small size should result in an overall simple solution path. But this is only true, if the "simplicity" is determined by a single and distinct step selected as the "most complex" one. As soon as more than one step enters into the calculation, *simplest first* cannot deliver the optimal score. This is the case for *any* scoring model one might choose, and of course for the BERT scoring.

9.2 ONE-STEP LOOK AHEAD

When analyzing some solver generated or manual resolutions paths one easily can notice a typical effect that occurs frequently. A number of eliminations are performed but there seem to be no substantial progress. Then almost all of a sudden an elimination hits the right point and a series of simpler eliminations follow. This may recur several times.

Looking deeper into the subject we find that an elimination has two completely different implications. One is of course clearing one or more targets,

[1] see http://en.wikipedia.org/wiki/Strategy

the other is enabling follow-up eliminations that were not possible before clearing the targets. The latter is indeed the more important one. An elimination that has no follow-up in the current resolution state is not urgent and can be kept in reserve until needed. This is the idea of *one-step look ahead*.

The execution of eliminations on demand clearly disrupts any predefined order of methods. If the effect would be only some reshuffling of the moves, the strategy is not worth discussing. But in almost all cases some eliminations become completely redundant, others are superimposed by *single eliminations*. This has severe implications on how to approximate the minimal score of a Sudoku problem.

9.3 POSITIONING ISSUES

In the previous chapters (and in traditional solving too) eliminations are considered to be relative to a particular resolution state. They seem to pop out of the blue and found by more or less tricky methods. This is a very limited view. With changing the order of the elimination sequence inside a resolution path we arrive at new problems. Placing an elimination at a different position creates a number of issues.

If the elimination is delayed, i.e. placed past the original position in the resolution path:

- (1) There may be less elimination candidates left. If none of them remains, the pattern becomes obsolete.
- (2) The elimination pattern is reduced by some cleared candidates of the original pattern.
- (3) The pattern is partially dissolved by candidates with value one.

If the elimination is shifted to a position earlier than the original:

- (4) The pattern may be not be ready to work.
- (5) There may be more elimination candidates compared to the original.
- (6) There may be extra but not harmful candidates in the pattern.

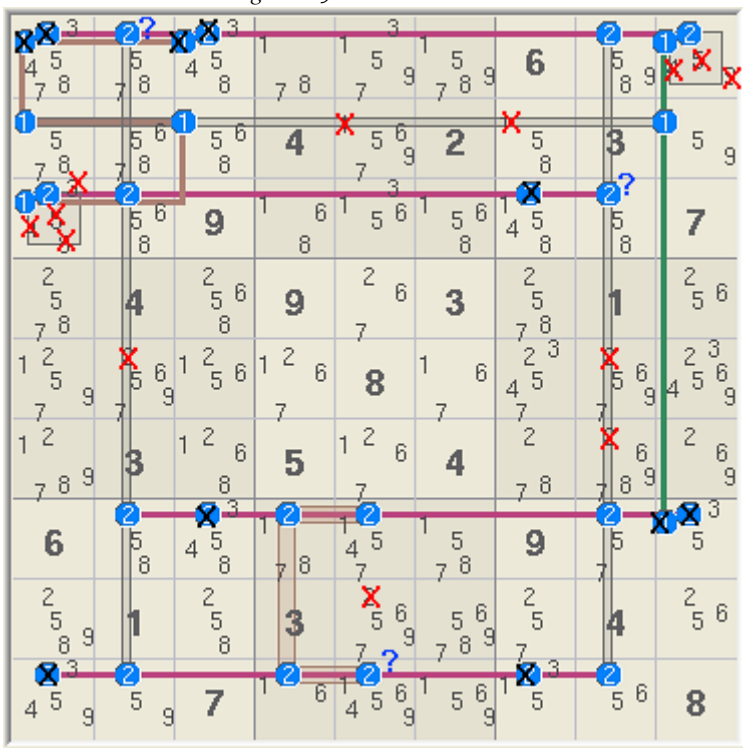
These situations are covered with the BERT notation by the same description, at least to some extent. Now it becomes apparent that the notation is also designed to support flexible positioning of eliminations. This property is a precondition for performing optimizations of resolution paths in a convenient way. All parts have stable identifiers and are well defined from the start.

9.4 POSITION INDEPENDENCE

To support the *look ahead strategy* effectively all eliminations will be transformed to a pattern that is relative to the beginning of the resolution path. For pragmatic reasons this is the resolution state one. This way all eliminations become absolute in this special way.

The consequences are explained with the following examples. At first we take diagram 5 mapped to state one.

Diagram 25: Absolute Matrix



000000600000402030009000007040903010000080000030504000600000900010300040007000008

This matrix is clearly not ready at resolution state one, but requires all 10 candidates marked with a black cross to be cleared before.

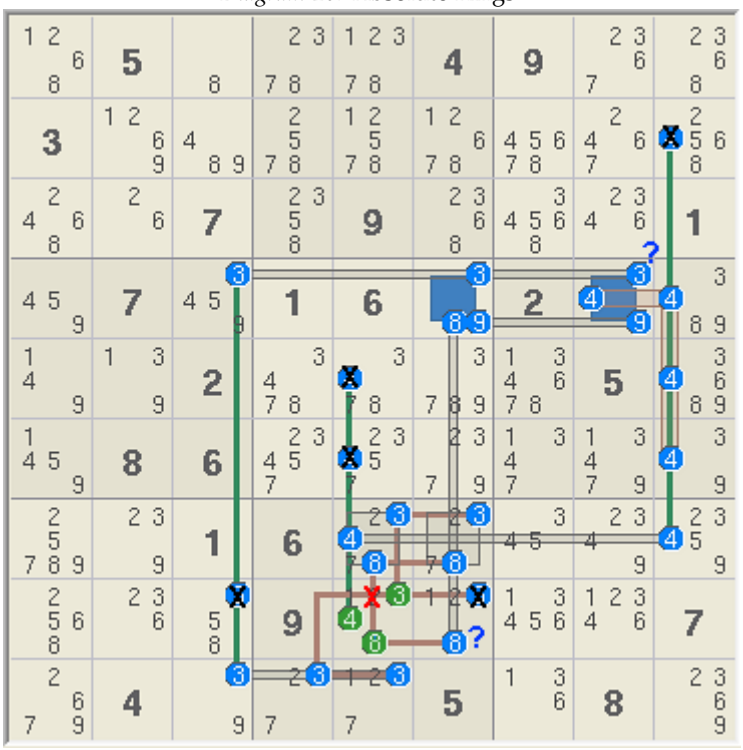
$$(12)11 + (12)13 + (12)79 + 237 + 273 + 29(17) = 0$$

The above equation is called *trigger equation* of the matrix. It represents the condition for the earliest position in the resolution path where the elimination can be applied. The three candidates 212, 238, 295 marked with "?" are already cleared in diagram 5, but are not part of the trigger equation. Their status does not have any impact except some score variation. The notation of the matrix expression remains the same, but has some more elimination candidates in the result term.

$$\begin{aligned}[2R1379, 1C9, 1B1 \quad &| \quad 2C28, 2B8, 1R2, R1C9, R3C1 \,] \\ &= \quad [252 + 285 + 2(56)8 + (459)19 + 12(57) + (3458)31]\end{aligned}$$

Eliminations calculations on top of a base equation behave slightly different. The next pattern shows diagram 12 mapped on resolution state 1.

Diagram 26: Absolute Rings



050004900300000000000709000107016020000200005008600000000160000000090000704000508 0

In this case the trigger equation is

$$229 + 4(56)5 + 38(36) = 0$$

and belongs to the combination of base calculation and elimination calculation. A trigger equation for a base equation alone makes no sense.

There is one elimination candidate at 285 and two candidates 348, 886 that are already cleared in diagram 12, but these are *not* trigger candidates. The result expression remains the same.

$$\begin{aligned}[((& R4C6, 3C3, 388B8 \mid 8C6, 3R49, R7C6) \\ & , R4C8, 4C559 \mid R7C5, 9C4, 4B6, 4R7) \\ & \qquad \rightarrow R8C5] \quad = \quad [285]\end{aligned}$$

Any elimination is accompanied by a trigger equation. The elimination is ready to fire, if all trigger candidates are cleared.

9.5 ELIMINATION LIFE-CYCLE

The BERT notation is designed to be invariant to the actual position in the resolution path. This allows to calculate the trigger candidates and the target candidates from the notation expression before the elimination is applied. There may be slight changes of the score value, if some of the candidates are cleared formerly.

The life-cycle of an elimination begins in a pending state. The trigger equation defines the earliest point where the elimination can be executed. Of course the trigger equation may be empty. Whether or not the elimination is used depends on the strategy.

There are three ways to end the life of an elimination

1. The elimination is executed.
2. All possible targets are already cleared.
3. A base equation is resolved.
4. A link equation becomes obsolete.

The last two conditions are about the *identity* of eliminations and their components. We say that the identity is lost, when one of equations is becoming trivial. This a pragmatic definition. In that case a pattern disintegrates into smaller parts, if not completely. The decay pattern can be still complex but is regarded as a different pattern than the original. It should be noted that a base equations or an elimination equation always remain valid regardless how many candidates of the pattern are assigned to a definite value.

10 OPTIMIZATION

The absolute optimal BERT resolution path is the one with the lowest possible score. This ideal result can only be obtained by checking all possible cases. This is practically impossible. It is a "hard" problem and no formula or shortcut can do the job. So the only chance is to approximate the lower bound of the score.

Optimization is different from strategy that tries to answer the question "what to do next" in a particular situation. Optimization rather takes an analytic top-down view on Sudoku. And this means far more than reducing the redundancy of a resolution path by deleting unnecessary steps combined with rearranging the order of steps. A number of alternative steps should be considered too. They may stem from several solving attempts of the same or of different solvers.

10.1 FIRST RESULTS

The question asked in section 1.4.4 is now ready for an answer. The example is simple enough to evaluate all possible combinations of steps. The lowest BERT score is 8 points. The corresponding resolution path:

```
[48B7|R7C3,R8C2] [7B7|7R9] [6B9|6R9] [6B7|6C1]
```

With only *group singles* (*box-line*) the resolution path needs 14 steps with one point each.

```
[2C9|2B3] [2C1|2B7] [3C9|3B3] [3R1|3B1] [1C1|1B7] [5C1|5B1] [5R1|5B3]
[5B8|5R7] [9C9|9B9] [9R9|9B7] [6C1|6B7] [1R9|1B9] [4C9|4B9] [6R9|6B9]
```

So the first path is the winner. I would like to avoid the term "better". It is merely the path with lesser BERT function points.

10.2 CONCEPT

The starting point of the optimization is a collection of eliminations that can occur at some resolution state of a particular puzzle. The collection must of course contain at least one complete elimination sequence. Some eliminations may share common parts in form of base equations. So the optimization is not aiming at the absolute optimum, but works in the scope of an explicit collection. To come near to score optimum of a puzzle, the *one-step look ahead strategy* is used. This means that for each resolution state a number of eliminations exists that are ready to fire. The readiness of an elimination is determined by an associated trigger equation.

The *one-step look ahead* puts all ready eliminations on hold unless their targets enable at least one more elimination. The others are waiting. This applies not for *simple single* eliminations for performance reasons. These are executed greedily and all the others on demand. Although this disrupts the order of moves, it does not alter the total score.

There is a prototype implementation of an optimizer that

- translates elimination notations to level 1
- checks for obvious errors

- checks for completeness
- calculates scores
- allows stepping through choices manually
- steps through all choices automatically

The automatic mode works with a modified *Dijkstra algorithm*[1] to find the shortest path in a graph. The nodes are the resolution states. Unlike the original algorithm the nodes are creates on demand. This is mandatory because the node distances depend on previous nodes. The algorithm requires naturally a lot of memory resources.

10.3 INPUT

The current implementation of my optimizer requires input as below. The main part of this input is written in BERT notation. Machine readable optimizer input is the central motivation for that notation.

The following example is about a reasonably difficult puzzle ($SER = 9.1$). The main path is generated by *Xsudo* and shown in full length. Some manual changes are made and the ALT sections contain selected steps taken from other solvers.

[1] *Dijkstra Algorithm* https://en.wikipedia.org/wiki/Dijkstra's_algorithm

```
$$BERT V1
$SUDOKU=050037008000061350309000000000310200105790030003400001960070003004003900530009010
$PATH=XSUDO
# step 1+2 group single
# step 3 (two string kite)
XS3=[(5R4,5B9|5C9)->5C6,5R7]
# step 4 (single ALS logic)
XS4=[(R12C4,1C24,8R3|89B2,1R8,R3C2)->R7C4,2C4]
# step 5 (DN-loop grouped) covered by step 7
#XS5=[(7R3,15R7,1C2,59C9|R3C2,R7C4,R4C9,1B7,5B9)->R2C9,7B3]
# step 6 group single
# step 7 (looped chain)
XS7=[(2*9C9,9C8,5C9,1R17,4B3,5B8|1C3,R1C78,5R8,R7C4,R4C9,9B6)->R2C9]
# step 8 (chain)
XS8=[(2*4R3,58R3,24R5,R1C7,1R3,4R9|4C59,R3C245,R5C2,14C7)->R3C6,2C6]
# step 9 (rank 3 logic)
X9A=((2*5R7,5R68|5C57,5B9),2*1C4,16C3,26C6|R1C3,R56C6,R7C4,6R4,1R7)
XS9=[X9A->R8C4,2B8]
# step 10 (rank 3 logic)
XS10=[(X9A,6C4|R8C4)->R9C4,2B8]
# step 11 group single
# step 12 (rank 4 logic)
X12A=(2*R1C7,R1C8,R2C9,2C9,4R9|249B3,4C7,R9C9)
X12B=(X12A,2*2B8,R6C5,5C7,R7C4,1C3|2C5,5R67,1R17,2R8)
XS12=[(X12B,4R79|R7C6,4B9)->R9C5,8B8,8C5]
#[(((2*R1C7,R1C8,R2C9,2C9,4R9|249B3,4C7,R9C9),2*2B8,R6C5,5C7,R7C4,1C3|
   2C5,5R67,1R17,2R8),4R79|R7C6,4B9)->R9C5,8B8,8C5]
# step 13 (rank 5 logic)
X13A=((2*R7C6,R7C8,R9C479|8B8,8R7,2B9,67R9),2*R7C7|4R7,4C7)
X13B=(2*R5C7,R5C6,X13A|8R5,8C7,2C6)
XS13=[(2*R5C9,R389C9,X13B|276C9,5B9,6R5)->4C9,4B6]
# step 14 (rank 4 logic)
X14A=(2*5B5,5C7,8C5,5R8,4R3-5R6,R68C5,5B9,R3C5)
X14B=(2*1R3,6R38,4R9,R1C7,R5C9|R3C7,14C7,6C89,4C9)
XS14=[(X14A,X14B,1R8,4R3|1C2,R8C4,4C5)->R3C6,5C6]
# step 15 group single # step 16 (rank 3 logic)
XS16=[((2*R6C5,R8C5,R4C6,5C9|8B5,8C5,5R48),2*2B4,6B4|6R4,2R6,R6C1)->2C2,2R8]
XS17=[(2*1R7,1R8,5R67,6C36,2C6,1B1|1C24,R1C3,R7C4,R56C6,5C7,6R4)->R7C3,2R7]
XS18=[(2R57,2C9|2C6,2B9)->2C2,2R3]
XS19A=(R3C2,1R8|1C2)
XS19=[(8B2,XS19A|8R3)->R8C4,8C4]
XS20=[(8R2,R9C4,XS19A|8B1,8C4)->R8C4,6C4]
XS22A=(6R8,2R7,8C8|R78C8)
XS22=[((2*8R5,6R5,XS22A|R5C7,8B6,6C9),2*2R5|R5C6,2C6)->R5C2]
XS24=[(2R157,9R1,4C8|R1C48,R7C8,2C6)->2C2,2B1]
XS26=[((2*5B9,5B6,9C9,R1C8,4R7|R7C7,5C7,R4C9,9B3,4C8),6R8,2R7-R8C9,R7C6)->R8C8,2C8]
XS27=[(2*5C6,6C6,5C9,28R5,XS22A|R4C6,5R4,R5C26,R8C9,8B6)->R6C6,2C6]
XS28=[(((2*R6C5,8C7,8B8|8R67,8C5),2*2C6,R5C2|28R5,2B5),4R7|R7C6)->4B9,R9C7]
XS29=[(R569C7,R5C9,5C9,6R8|78C7,R8C9,45B6)->6C8,6B6]
XS30=[(2*1R1,6R16,8R9,5R6,7C7,R7C3|R1C3,R6C67,R9C7,18C3,6C1)->1C7,R3C7]
XS_P33={R12C4|29C4} # (naked pair)
XS35=[(45C7,R5C9|4R5,R7C7)->R6C7,6B6]
XS36=[(R39C7,6C8|7C7,6R3)->R8C8,8B9]
XS37=[(R7C4,5C7,8C8|58R7)->R6C7,8B6]
XS38=[(R5C9,4C7,5B9|4R5,R7C7)->R8C9,6C9]
```

```
XS40=[(R6C7,2R3,7C8,5C9|57B6,R3C8)->R8C9,2C9]
XS41=[(R58C2,2R68|28C2,2C5)->R8C1,7R8]
XS42=[(R6C7,5C9,7R8|5B6,R8C9)->7C2,7R6]
XS43=[(2*5R4,5R6,6C6,6B6|R4C9,5B6,6R5,R6C6)->R4C6]
XS44=[(R6C7,25C9,6R3,7B9|R389C9,5B6)->R3C7,7C7]
XS46=[(R8C9,57C7|R6C7,5B9)->7B9]
XS_P47=[R9C59|24R9] # (naked pair)
XS_P49=[R7C4,R8C5|58B8] # (naked pair)
XS50=[(R5C2,8R8,2B5|2R5,8C2)->R6C5,8C5]
XS52=[(8R4,9R6,7C8|R46C8)->R6C2,8B4]
XS54=[(R2C34,R4C39|2R2,7C3,6R4)->9R2,9C9]
$ALT=SUEX
SE8=[(4C6,4C8,2B3,24R5|R5C2,R1C8,4B6,4R7)->R3C6,2C6]
SE9=[(1R7,1B1,8R3,R12C4|1C3,R3C2,8B2,9C4)->2C4,R7C4]
SE10=[((3*1C4,5R67,5B8,16C3,26C6|2*R7C4,5C57,R1C3,R56C6,6R4,1R7),6C4|R8C4)->2B8,R9C4]
$ALT=OTHER
LX8=[(4C68,4B4,2R5,29B3|4R47,R5C2,R1C8,R2C9)->2R3,2C6,R3C6]
LX13=[(146C7,R5C9,56B9|R137C7,4R5,R8C9)->6B6,6C8]
LX18=[(8R8,2B7,R5C2,2C6,R6C5|R8C1,8C25,2R5)->2R9,2C5,2B8]
HO20=(1B1,8R2,R3C2,6C36,2R5,5R67|R1C3,8B1,6R4,R5C6,2C2,R6C6,5C7)
HO21=((HO20,2*1C4,5B8|2*R7C4,5C5),1R8|1C2)
HO22=[HO21->8C4,R8C4]
HO23=[(HO21,R9C4|8C4)->6C4,R8C4]
```

The file structure is pretty simple. The first line assures that the content is related to BERT. Comments start with '#'. The following line contains the Sudoku puzzle. Then there are some names sections or the type "PATH" or ""ALT". Any PATH section must contain a full path with a sequence that solves the puzzle completely.

For convenience *singles* and *group singles* need not to be listed. These are automatically generated by the current optimizer implementation. Calculated base equations generate *equation singles* also if appropriate.

10.4 VERIFICATION

The input data on level 2 do not show any candidates intentionally, neither for the logic network nor for targets. To check the correctness of eliminations recorded in such a way, equations are translated into level 1. All components are shown with their candidates relative to resolution state one. Trigger and targets can be verified to match the original pattern.

The log output of the verification phase of the optimizer is shown for steps *X9A*, *XS9* and *XS10* as an example.

```
$ SRC _006=(2*5R7 5R68|5C57 5B9)
+ ROW 5R7=(574*,576,577)
+ ROW 5R7=(574*,576,577)
+ ROW 5R6=(565,566,567*)
+ ROW 5R8=(584,585,589*)
- COL 5C5=(535*,565,585)
- COL 5C7=(567*,577)
- BOX 5B9=(577,589*)
= EQU _006=(566,574*,576,584)  VAL=9
  EQS [_006|*]=[534,174,274,874,576,577,584,585] CORE=(574*) TRIG=[566,576,584] VAL=9
$ SRC X9A=(_006,2*1C4,16C3,26C6|R1C3,R56C6,R7C4,6R4,1R7)
+ EQU _006=(566,574*,576,584)  VAL=9
+ COL 1C4=(174,184*)
+ COL 1C4=(174,184*)
+ COL 1C3=(113,173*)
+ COL 6C3=(613*,643)
+ COL 2C6=(236,256,266,276*)
+ COL 6C6=(646,656,666*)
- NUM R1C3=(113,213,613*)
- NUM R5C6=(256,656,856*)
- NUM R6C6=(266,566,666*,866)
- NUM R7C4=(174,274,574*,874)
- ROW 6R4=(641*,643,646,648,649)
- ROW 1R7=(173*,174)
= EQU X9A=(236,276*,576,184*,584)  VAL=28
$ SRC XS9=[X9A->R8C4,2B8]
+ NUM R8C4=(184*,284,584,684,884)
+ BOX 2B8=(274,276*,284,285,294,295)
- EQU X9A=(236,276*,576,184*,584)  VAL=28
= TGT XS9=[284] CORE=(276*,184*,584) TRIG=[236,576] VAL=32
$ SRC _007=(X9A,6C4|R8C4)
+ EQU X9A=(236,276*,576,184*,584)  VAL=28
+ COL 6C4=(684,694*)
- NUM R8C4=(184*,284,584,684,884)
= EQU _007=(236,276*,576,694*)  VAL=32
$ SRC XS10=[_007->R9C4,2B8]
+ NUM R9C4=(294,694*,894)
+ BOX 2B8=(274,276*,284,285,294,295)
- EQU _007=(236,276*,576,694*)  VAL=32
= TGT XS10=[294] CORE=(276*,694*) TRIG=[236,576] VAL=35
```

The input data and results are translated to level 1 and listed for each expression and sub-expression. Nested expressions will be decomposed recursively. Implicit sub-expressions get automatically generated names beginning with "_". Candidates marked with "*" are solution values. The score values are calculated with all candidates of the pattern that are live at state one. So these values are the maximum possible score values for the patterns.

The base equation *_006* creates an additional *base equation single, _007* does not because it contains two solution values.

The log output is sufficient to check the notation manually for correctness. There are limited internal plausibility checks also.

10.5 OPTIMIZATION RESULT

As the main path is generated by Xsudo with it's own concepts to choose the order of steps. So one cannot expect a good BERT score from the start. Manual changes had been made dropping obviously unnecessary steps and for the usage of shared parts to demonstrate the effect of re-use. This results

in a total score of 699 points. The optimizer finds a path of 536 points by reshuffling and dropping eliminations. If the elimination set is extended with a few patterns from other solvers, a path with 495 points is achieved. Although this seems to be a good result there is no way to be sure about.

```
GRP [4B2|4R3]=[432,437,438,439] CORE=(435,436*) TRIG=[] VAL=1
TGT XS3=[576] CORE=(546*,577) TRIG=[] VAL=6
TGT XS7=[229,729] CORE=(429*,929) TRIG=[437-,438-,439-,576-] VAL=25
TGT LX8=[236] CORE=(436*,238*,239,256) TRIG=[438-] VAL=22 (-1)
TGT XS4=[274] CORE=(214*,224,174) TRIG=[] VAL=18
TGT XS9=[284] CORE=(276*,184*,584) TRIG=[236-,576-] VAL=32
TGT XS10=[294] CORE=(276*,694*) TRIG=[236-,576-] VAL=7 (-28)
GRP [6B1|6R1]=[617,618] CORE=(611,613*) TRIG=[] VAL=1
TGT XS12=[895] CORE=(865,874,295,495*)
     TRIG=[617-,618-,229-,729-,274-,284-,294-] VAL=51
TGT XS14=[536] CORE=(436*,546*,566) TRIG=[617-,432-,437-,438-,439-,895-] VAL=49
GRP [5C6|5B5]=[565] CORE=(546*,566) TRIG=[536-,576-] VAL=1
TGT XS17=[273] CORE=(173*,276*) TRIG=[236-,565-,576-] VAL=28
TGT XS18=[232] CORE=(239,252*) TRIG=[229-,273-,274-] VAL=10
GRP [7B3|7R3]=[732] CORE=(737,738,739*) TRIG=[729-] VAL=2
TGT XS20=[684] CORE=(184*,694*) TRIG=[232-,432-,732-,294-] VAL=14
2 SINGLES [894,697,699]
TGT XS27=[266] CORE=(566,666*,276*) TRIG=[536-,273-,274-,576-,684-] VAL=30
TGT XS19=[884] CORE=(824,184*) TRIG=[232-,432-,732-] VAL=8 (-3)
TGT XS22=[452] CORE=(252*,852) TRIG=[273-,274-,684-] VAL=19 (-6)
TGT XS28=[497] CORE=(477,478*,897) TRIG=[236-,452-,565-,266-,884-,894-] VAL=27 (-1)
TGT LX13=[648,668] CORE=(657,659*,667,688*) TRIG=[497-,697-,699-] VAL=20 (-2)
TGT XS30=[137] CORE=(117*,737) TRIG=[617-,618-,565-,668-,273-,894-,895-] VAL=25
7 SINGLES [113,417,832,873,174,182,584]
GRP [4R5|4B6]=[448,449] CORE=(457*,459) TRIG=[452-] VAL=1
TGT XS16=[282] CORE=(252*,262,285) TRIG=[565-] VAL=27
TGT XS36=[888] CORE=(688*,897) TRIG=[618-,137-,437-,648-,668-,497-,697-] VAL=9
TGT LX18=[295] CORE=(265*,276*,293)
     TRIG=[236-,452-,565-,266-,273-,282-,884-,888-] VAL=17
4 SINGLES [435,836,476,499]
MATRIX SE_P32=[218,618-,229-,729-] TRIG=[417-,437-,438-,439-]
     CORE={418,918*,429*,929} VAL=5
EQS [_030|589*]=[549,577,584-,585,289,689,789] CORE=(589*)
     TRIG=[218-,618-,476-] VAL=15
9 SINGLES [824,234,534,235,835,638,646,846,566,667,767,867,874,288,788]
TGT XS44=[737] CORE=(637*,767-,797*) TRIG=[229-,638-,667-,867-,788-] VAL=13 (-4)
21 SINGLES [211,221,222,722,823,738,239,639,742,843,749,852,256,657,
     261,262,762,865,876,278,781,881,882,285,293,793,897,799]
TGT XS54=[929] CORE=(924*,949*) TRIG=[823-,824-,843-,449-,549-,749-] VAL=12
26 SINGLES [611,213,914,418,421,821,422,723,224,441,741,841,842,942,
     643,748,948,649,656,857,459,661,761,862,866,868,968,477,878]
OPTIMAL SCORE: 495    TOTAL_NODES: 37632
```

The number of eliminations shrinks to 26, where *singles* are not counted. Four alternative eliminations had been used ($LX8, LX13, LX18, SE\_P32$). Candidates marked with "-" are already eliminated at that point (e.g.all trigger candidates). Negative values enclosed in parenthesis show the difference to the maximal score. For $XS10$ this difference is large because of re-use of a base from $XS9$. Look at the corresponding input notations.

This example shows that an optimized BERT score is by far lower than the score of paths generated from a static method orders.

11

MINIMALITY

This additional chapter is concerned with minimal properties of puzzles. Although not part of the BERT framework, the same math model is used. So there is some connection. The importance of minimal candidate arrangements is discussed already in the previous chapters. There are more interesting aspects of minimality.

11.1 MINIMAL GIVENS

In a Sudoku puzzle with minimal givens none of them can be calculated from the others. If the Sudoku has a unique solution, all problems with one given less have multiple solutions. This means that uniqueness is strongly related to minimality.

There had been a lot of investigations about minimal givens. The many results are not repeated here.

11.2 MINIMAL EQUATIONS

The question here is whether all of the 324 defining equations of a (minimal) Sudoku are necessary to assure that there is a unique solution. The answer is clearly NO! And it seems that this is the case all the time. This is equivalent to ask for dropping equations and still retaining the number of solutions.

If we take an arbitrary resolution path of a particular puzzle with a unique solution, some of the 324 equations do not appear in any of the elimination steps. This means they are not necessary to fix the values of all candidates. The natural follow-up question is: what happens if we try drop more equations from the set of used ones. We will soon come to a situation where we are unable to find explicit elimination steps that solve the puzzle. But we can switch to a recursive solver and perform an uniqueness check on subsets of the 324 native base equations. Reducing the equation set step by step we finally arrive at a configuration where dropping any of the remaining equations results in multiple solutions of the puzzle. The minimal equation set together with the given candidates force all other equations to be satisfied also. It's obvious that each candidate must occur in at least one equation. Otherwise uniqueness is impossible.

An investigation of minimal equations sets of a few dozen puzzles produced the following completely unexpected results. Starting from stage one, a little program selects randomly equations and performs uniqueness checks. If the remaining set has still a unique solution, the selected equation is dropped. This loops until a minimal set is reached. Such Monte Carlo style procedure is repeated several times (typically 1000) to generate different minimal sets.

An example puzzle with the associated list of 139 minimal equations follows. The puzzle has minimal given candidates and is derived from the example of the previous chapter. The 24 givens leave 57 cells open. The direct eliminations by the givens relate to $4 * 24 = 96$ base equations. These are not subject of the reduction, because the process starts at stage 1. At this point there are 222 open candidates and $4 * 57 = 228$ nominal base equations. Some redundant box equations exist, but this is no issue for the reduction. Finally we have $96 + 139 = 235$ equations that lead to a unique solution and also forcing the remaining $324 - 235 = 89$ equations to be satisfied.

```
000007008000061050309000000000010200105090030003400001060070003004003900500000010
456237198782961354319854627647315289125798436893426571961572843274183965538649712
```

```
R1C1458,R2C12349,R3C56789,R4C12689,R5C2,R6C12567,R7C13467,R8C489,R9C3579
1R38,2R1359,3R1,4R1249,5R1678,6R134569,7R248,8R24589,9R14679
1C237,2C235689,3C247,4C56789,5C245679,6C369,7C123478,8C12567,9C248
1B18,2B12457,3B5,4B129,5B26,6B589,7B369,8B245679,9B368
```

11.2.1 *Number of Equations*

The number of redundant equations was expected to be small, but in fact the number is much larger, varying between 80 to 100. That means there is a huge redundancy in the original Sudoku puzzle statement, even if the puzzle has minimal givens. At this point it remains unclear what kind of consequences this might have.

11.2.2 *Correlation with Severity*

The expectation that puzzles with high severity (SER rating value) come with a larger set of minimal equations was not met. There seem to be no significant difference for all cases investigated. Even more, some puzzles with *single* solutions only needed more equations that others with $SER = 10$.

Reducing the equation set indeed affects the severity of the restricted puzzle itself. For a formerly easy to solve puzzle it's extremely hard to find an elimination at all after minimalization. Unfortunately there is no solver that can handle reduced equation sets. So the question remains open how exactly the severity is altered.

11.2.3 *Common Equations*

One can expect that all minimal equation sets of the same puzzle have some common intersection. This is not the case usually. The minimal sets seem to be very flexible instead. Even if we deliberately exclude a dozen of equations, there are still plenty of minimal sets with no common intersection.

More and thorough investigation is needed to understand minimal equation sets.

11.2.4 *More Details*

Next we can mark some of the equations as *base equation* only. This means that only marked equations are able to set candidates to value 1 during the solving process. It is obvious that the number of base equations is at least $81 - \#\text{givens}$, because any candidate with value 1 requires a different base equation. Now we minimize the base equations in Monte Carlo style. The surprising result is that the actual number of minimal base equations is near the lower limit. Again this result seems to be independent of the puzzle severity (SER).

The minimal base equation configuration does not match the total equation minimum, but seems to require more link equations. There are more detail questions to ask, but the above issues alone have no really convincing interpretation. All in all more questions are raised than answered.

12

SUMMARY

There are many ways to deal with Sudoku puzzles. The view that is expressed here emphasizes properties rather than methods or procedures. Although BERT is not "the" Sudoku theory, it has a very broad scope, a reasonable degree of abstraction and the potential for further refinement. The BERT concept is not yet another solver, it does not solve puzzles at all. BERT is a *resolution theory* that exactly defines which kind of eliminations are fitting into the concept. This determines the scope. It analyzes existing resolutions and their eliminations regardless of their origin. The BERT concept works on basis of very few construction principles and does not depend on what is traditionally called "method".

A leading principle is "straightforward logic with no assumptions". This imposes some restrictions on eliminations to be used, but that looks harder than it actually is. Most eliminations that use contradictions can be transformed into equivalent straightforward logic.

The BERT concept consists of two main parts. The first comes with a formal description of a resolution path and its components. A definite algorithm verifies the correctness of a description. The second part attaches a metric that allows to compare resolution paths partially or as a whole. The score value given by the metric is of course only one of the many complexity aspects the puzzle might have. The score value also relies on some general principles and has a clear meaning.

A brief list of the main ideas follows.

- The Sudoku puzzle is redefined as a system of 324 arithmetic equations with integer variables (candidates) of value 0 or 1. Each equation for rows, columns, boxes of a number and for cells has the value 1.
- The main idea of an abstract definition of eliminations is the *sub-puzzle* as a minimal set of equations.
- The *pattern* definition restricts sub-puzzles further to those that confirm $Z = 0$ or $B \geq 1$ or $L \leq 1$ (Z, B, L any sum of variables) for all solutions of the sub-puzzle. Note that this is different from the term "pattern" used elsewhere and denotes an individual arrangement of candidates.
- An equation $B \geq 1$ is a *base equation* and a generalization of a "strong link".
  An equation $L \leq 1$ is a *link equation* and a generalization of "weak link".
  An equation $Z = 0$ is a *zero equation* and denotes one or more elimination targets.
- BERT patterns can be calculated by applying special operations on base and link equations. The operators are well defined and use arithmetic additions and subtractions of candidates and additionally some reductions.
- An appropriate notation of eliminations and of intermediate calculation steps is provided. Steps describe a part of an elimination and may be re-used in later calculations. More complex elimination patterns require a description with nested expressions. Only equation variables appear in any expression notation, but a transformation into candidates is always possible.

- For each elimination pattern there is a score value that is calculated from the notation in combination with a resolution state where the pattern is applied. The score value counts the total number of additions and subtractions performed by the verification algorithm of the pattern.
- The score of a resolution path is calculated by adding the scores of all steps. Re-used parts are counted only once, emphasizing the benefit of re-use. Different paths of the same puzzle can be compared by their score value.
- Each elimination pattern has a life cycle that can be calculated from the notation. A zero equation containing the trigger variables marks the earliest point, where the elimination can be applied. An elimination pattern is dead with no targets left or by destruction of one of its defining parts.
- The life cycle information allows to re-shuffle the order of steps of a resolution path or mixing several paths or parts of it. This is the prerequisite for constructing a resolution path with a lower total score value.
- The central strategic idea is to apply only elimination steps that enable at least one follow-up step. This way the resolution path takes the form of a directed graph that reflects the intrinsic dependencies of the path.
- An optimizer can calculate the minimal score of resolution paths constructed from a limited pool of steps. A calculation of the absolute score minimum of a puzzle is not realistic.

13 FINAL REMARKS

It's just a start, far form perfect.

Does it make Sudoku easier or more complicated? Yes and no.

BERT adds a new dimension. Solving alone is not enough. Finding efficient resolutions paths requires to compare and reshuffle lots of paths. This aspect makes Sudoku clearly more complicated. On the other hand BERT notation of complex elimination steps are more structured and understandable in my view.

13.1 OUTLOOK

The BERT concept is immediately applicable to other Sudoku variants, with diagonal base equations, with other shapes of boxes, other sizes.

Generally any problem statement that can be defined by base and link equations is BERT compatible. Whether or not this is an useful approach for an arbitrary problem statement of such kind is unknown.

The eight queens puzzle is a good example, if some additional restrictions assure a unique solution.

13.2 MISSING THINGS

First of all the lack of mass data makes quantitative or statistical statements impossible. All examples shown are produced manually. This is tedious even when using some support tools. The output of existing and available solvers is either not machine readable or not in a form that allows easy translation into BERT notation. Solvers are usually not designed for post-processing their output.

Automatic translation of AIC notation or DB braid notation seems (at least partially) manageable. A translation of the *SudokuExplainer* output seems hopeless. Solvers with graphical output only are even worse.

A generator for graphical presentation of steps (table and/or grid) or for the path graph is nice to have.

The optimizer allows to step manually through the path, with one-step look ahead at each state. The automatic mode finds the minimal score of a given elimination set. Advanced features like detection of re-usable parts are far away.

ABBREVIATIONS

BE	Def: Base Equation
BERT	Def: Base Equation Resolution Theory
NBE	Def: Native Base Equation
AB	Name: Alan Barker
DB	Name: Denis Berthier
GSF	Name: Glenn Fowler
SPF	Web: New Sudoku Player's Forum