APPROVAL PAGE FOR GRADUATE THESIS OR PROJECT

GS-13

SUBMITTED IN PARTIAL FULFILLMENT OF REQUIREMENTS FOR DEGREE OF MASTER OF SCIENCE AT CALIFORNIA STATE UNIVERSITY, LOS ANGELES BY

Benjamin James Bush
Candidate

General Mathematics
Field of Concentration

TITLE: Solving The Shirokuro Puzzle Constraint Satisfaction Problem With Backtracking: A Theoretical Foundation

APPROVED:	**Russ Abbott**	
	Faculty Member	Signature
	Anthony Shaheen	
	Faculty Member	Signature
	Gary Brookfield	
	Faculty Member	Signature
	Grant Fraser	
	Department Chairperson	Signature

DATE: ____________________

SOLVING THE SHIROKURO PUZZLE CONSTRAINT SATISFACTION PROBLEM WITH BACKTRACKING: A THEORETICAL FOUNDATION

A Thesis

Presented to

the Faculty of the Department of Mathematics

California State University, Los Angeles

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Benjamin James Bush

December 2011

PREFACE

This thesis documents much of the work that was done when I took an independent study course with Dr. Russ Abbott in 2008. Dr. Abbot is a professor of computer science at California State University Los Angeles. While traveling via airplane, Dr. Abbott discovered an interesting puzzle called “Shirokuro.” He thought that it would be interesting to try to solve the puzzle using computational techniques. Dr. Abbott later shared his interest with my colleague Alexandre Lomovtsev and me. We shared Dr. Abbott’s enthusiasm for the problem and began developing our computational Shirokuro puzzle solvers. The three of us met on a weekly basis for several months. Much progress was made. However, as with all interesting projects, there is always more work to be done. Any readers who are inspired by this thesis to continue my work should not hesitate to contact me; this research is ongoing. I am always eager to discuss new developments.

Benjamin James Bush

December 2011

ACKNOWLEDGEMENTS

There are many people who have aided me and supported me over the years. I would like to thank them, especially those who have guided me during the writing of this paper. I would like to thank Dr. Russ Abbott for introducing me to Shirokuro puzzles. His guidance, suggestions, and assistance in completing this work were invaluable. Additionally, I would like to thank Dr. Gary Brookfield and Dr. Daphne Liu, who helped me overcome some of the obstacles I encountered while working to understand some of the more difficult proofs in the literature. They also were instrumental in guiding me through the logistical aspects of the thesis submission process. I would like to thank Alexandre Lomovtsev for being my thoughtful colleague. He shared with me the discovery of the "X" pattern and its implications for solving Shirokuro puzzles (see section 6). Information on the history of the Shirokuro puzzle came from the staff of Puzzability. I would like to acknowledge Amy Goldstein and the staff of Puzzability for sharing the insight illustrated in Figure 7. I extend my greatest appreciation to my parents, Bruce and Alma Bush. They have provided love and support from the day I was born up until the completion of this thesis, and beyond. I also thank my wife, Rene Bush, for giving me the love and support that I needed to persevere until my thesis was complete.

The funding for this project was provided by the National Science Foundation under Grant # HRD-0331537 and the CSU Chancellor's Office.

ABSTRACT

SOLVING THE SHIROKURO PUZZLE CONSTRAINT SATISFACTION PROBLEM WITH BACKTRACKING: A THEORETICAL FOUNDATION

By

Benjamin James Bush

Shirokuro is a Japanese puzzle recently featured in Spirit magazine. It is played on a grid of n by n cells, each of which may be empty or contain a black or white disk. The goal is to fill every empty cell with a white or black disk so that the following conditions are satisfied: (1) No two-by-two region contains four disks of the same color. (2) Each pair of like-colored disks is connected via a chain of disks which travels horizontally or vertically through disks of the same color as the pair. We describe Shirokuro as a constraint satisfaction problem, and then propose a backtracking algorithm which solves the puzzle by incrementally building candidate solutions. We propose several relaxed constraints which may help to speed up the algorithm and give an overview of some advanced backtracking techniques.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

Shirokuro, which means “black and white” in Japanese, is an obscure Japanese puzzle which was featured in August 2008 edition of Spirit magazine. The gameplay of Shirokuro is similar to that of the much more popular Sudoku. Shirokuro is played on a square grid of $n \times n$ cells, each of which may be empty or contain a disk that is either black or white in color. The goal of Shirokuro is to fill in every empty cell with a white or black disk so that conditions specified in Figure 1 below are satisfied. Cells which initially contain a black or white disk are fixed and cannot be modified; however, cells that are initially empty must be modified by the player until a solution is found. An example Shirokuro puzzle along with its solution is illustrated in Figure 2 on the following page.

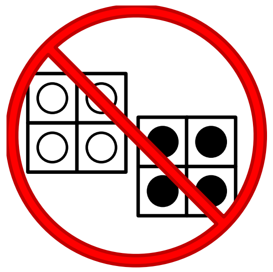
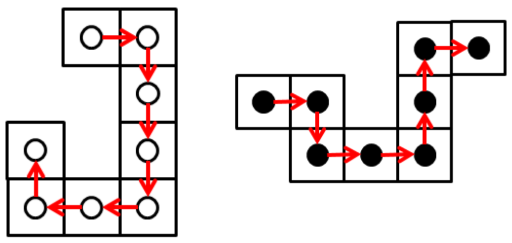
Condition	Illustration
No 2×2 region of the grid contains four disks of the same color (e.g. no “clumps” allowed).	
Each pair of like-colored disks is connected via a chain of disks which travels horizontally or vertically through disks of the same color as the pair.	

Figure 1: The conditions that must be met to solve a Shirokuro puzzle.

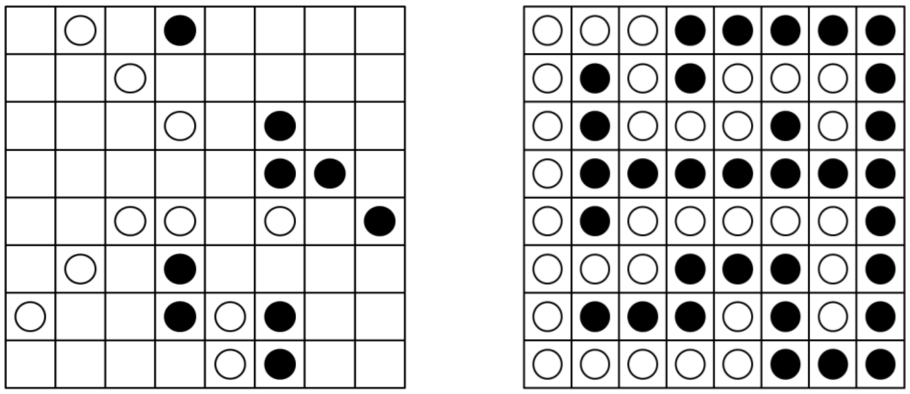


Figure 2: An example a Shirokuro puzzle (left) and its solution (right).

Shirokuro’s more famous cousin, Sudoku, has been formulated as a constraint satisfaction problem (CSP) [1]. We do the same for Shirokuro in chapter 2, where we define Shirokuro as a CSP.

While CSPs such as Shirokuro can potentially be solved by brute force enumeration, backtracking algorithms are more efficient [2]. A simple backtracking algorithm for solving Shirokuro puzzles is given in chapter 3. In chapter 4, we enhance the simple backtracking algorithm with several Shirokuro “tricks” which are implemented as relaxed constraints. Chapter 5 describes some advanced techniques that may further enhance the efficiency of the backtracking algorithm. Finally, we conclude with an outline of our planned future work and a summary of what has been accomplished so far.

CHAPTER 2

Shirokuro as a Constraint Satisfaction Problem

Many puzzles, such as 8 Queens, Instant Insanity, and Sudoku can be expressed as constraint satisfaction problems (CSPs) [3] [4] [1]. In this section we briefly discuss CSPs in general. We then express Shirokuro as a CSP. The regularities and properties that are common to all CSPs help us solve Shirokuro puzzles.

2.1 Constraint Satisfaction Problems

We loosely follow the definitions and notation given in section 2 of Baccus' "A Uniform View of Backtracking" [5], which the reader is encouraged to reference for a more in-depth and general understanding of CSPs. The fundamental elements of a CSP are a variable set X and a constraints set C. Each variable $x \in X$ may only be assigned values from a set called the domain of x, which we designate $Dom(x)$. We use the notation $x \leftarrow v$ to denote an assignment which assigns the value $v \in Dom(x)$ to the variable x.

An assignment set α is defined over a subset of the variable set X. This subset is called the scope of α, which we denote $Scope(\alpha)$. For example, if an assignment set α is defined over $Scope(\alpha) = \{x_1, x_2, \dots, x_k\}$, then α will be of the form $\{x_1 \leftarrow v_1, x_2 \leftarrow x_2, \dots, x_k \leftarrow v_k\}$, where each $v_i \in Dom(x_i)$. An assignment set may also be viewed as a function; $\alpha: Scope(\alpha) \longrightarrow \bigcup_{x \in Scope(\alpha)} Dom(x)$ where $\alpha(x) = v$ if the assignment $x \leftarrow v$ is in the assignment set α.

Like assignment sets, each constraint $c \in C$ is defined over a subset of the original variable set X, which is the scope of the constraint and is denoted $Scope(c)$. The purpose of a constraint is to place restrictions on what values can be simultaneously assigned to the variables in its scope. Each constraint can thus be thought of as a collection of allowable assignment sets, each of which simultaneously assigns a value to each variable in the scope of the constraint. Whenever a variable x is a member of $Scope(c)$, we may declare that the variable is constrained by the constraint c.

An assignment set α is said to activate the constraint c if α assigns a value to every variable constrained by c, i.e. if $Scope(c)$ is a subset of $Scope(\alpha)$. If α activates c and there exists an assignment set $\beta \in c$ such that β is a subset of α, then α is said to satisfy constraint c. On the other hand, if α activates c and no such member of c exists, then α is said to violate c. Note that if α does not activate c, then it neither satisfies nor violates c.

If an assignment set α satisfies all of the constraints that it activates, then α is said to be consistent. If on the other hand α violates one of the constraints that it activates, then α is called a *no-good*. If $Scope(\alpha) = X$, then α assigns a value to all variables in X and is known as a complete assignment set. Otherwise, it is known as a partial assignment set. A solution to a CSP is an assignment set that is both complete and consistent.

As an example of a CSP we borrow from Russell and Norvig [6]. Suppose we are given the task of coloring the states on the map of Australia red, green and blue in such a way that no two neighboring states are of the same color. This corresponds to the graph coloring problem with a the graph illustrated in the figure below and color set $\{red, green, blue\}$. It is straightforward to express this as a CSP. For the variable set, we have $X = \{WA, NT, Q, SA, NSW, V, T\}$. For each variable $x \in X$ we have $Dom(x) = \{red, green, \ blue\}$.

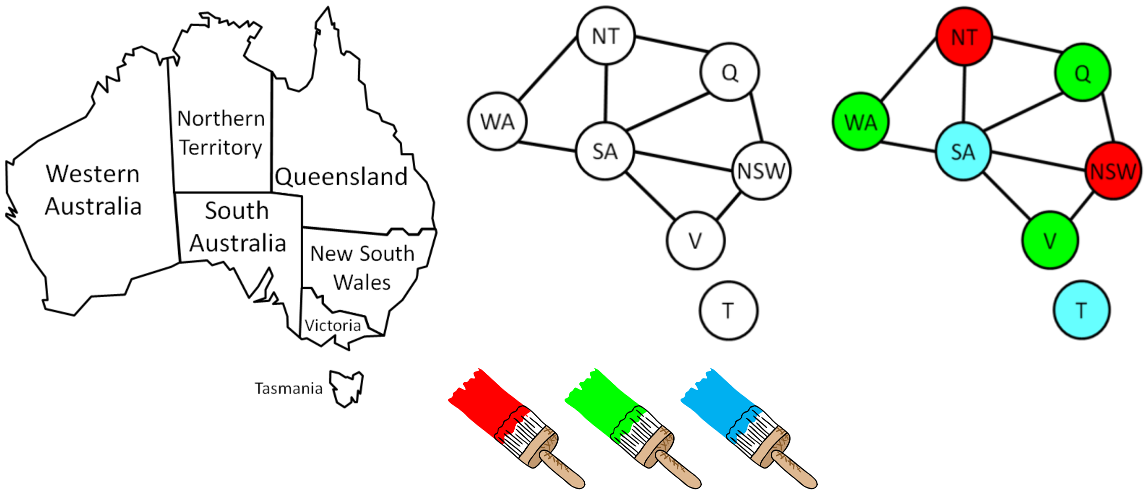


Figure 3: A map of Australia (left), its associated graph coloring problem (middle), and solution (right)

For each edge (x_1, x_2) in the graph we have the constraint:

$$\left\{ \begin{array}{c} \{x_1 \leftarrow red,\ x_2 \leftarrow green\}, \{x_1 \leftarrow red,\ x_2 \leftarrow blue\}, \{x_1 \leftarrow green,\ x_2 \leftarrow red\}, \\ \{x_1 \leftarrow green,\ x_2 \leftarrow blue\}, \{x_1 \leftarrow blue,\ z_2 \leftarrow red\}, \{x_1 \leftarrow blue,\ x_2 \leftarrow green\} \end{array} \right\}$$

This is more concisely expressed as:

$$\{\ \{x_1 \leftarrow v_1,\ x_2 \leftarrow v_2\}\ \mid\ v_1, v_2 \in \{red, green, blue\},\quad v_1 \neq v_2\}$$

An assignment set violates the above constraint whenever it assigns the same value to x_1 and x_2.

2.2 The Shirokuro Variable Set

For Shirokuro, we have the variable set X is defined as follows:

$$X = \left\{ x_{i,j} \;\middle|\; i, j \in \{1,2,3 \dots n\} \right\}$$

so that the variable $x_{i,j}$ corresponds to the cell that is located in the ith row and jth column. The domain of each of these variables is the set {●,○} where ● indicates a black disk and ○ indicates a white disk. There are times when it is convenient to graphically represent a Shirokuro assignment set. At such times we will display a Shirokuro grid such as the one illustrated below:

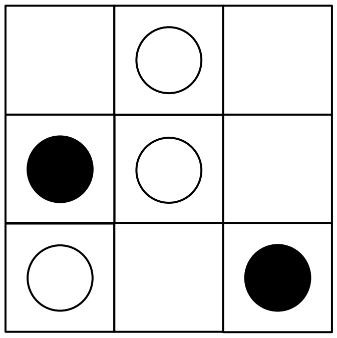


Figure 4: Graphical representation of an assignment set

This graphical representation corresponds to the assignment set:

$$\{x_{1,2} \leftarrow \circ,\ \ x_{2,1} \leftarrow \bullet,\ \ x_{2,2} \leftarrow \circ,\ \ x_{3,1} \leftarrow \bullet,\ \ x_{3,3} \leftarrow \bullet\}$$

so that the empty cells in the graphical representation correspond to variables outside of the scope of the assignment set. We will henceforth use the terms "cell" and "variable" interchangeably.

2.3 The *NoClump* Family of Constraints

We can express the condition "no 2×2 region of the Shirokuro grid may contain four disks of the same color" as a series of constraints. For every element of $\{(i,j) \mid i,j \in \{1,2,3,\dots,n-1\}\}$ we have the following constraint:

$$NoClump(i,j) = \left\{ \left\{ \begin{matrix} x_{i,j} \leftarrow v_1,\ x_{i+1,j} \leftarrow v_2, \\ x_{i,j+1} \leftarrow v_3,\ x_{i+1,j+1} \leftarrow v_4 \end{matrix} \right\} \ \middle|\ \begin{matrix} v_k \in \{\bullet,\circ\} \text{ for } k \in \{1,2,3,4\} \\ v_p \neq v_q \text{ for some } p \text{ and } q, \\ p \neq q, \quad p,q \in \{1,2,3,4\} \end{matrix} \right\}$$

so that the constraint $NoClump(i,j)$ is the set of all assignment sets that do not simultaneously assign the same value to all 4 variables of the scope corresponding to the 2×2 region of cells at position (i,j), i.e. there is no "clump" at position (i,j).

2.4 Connectivity Graphs and the *TwoComponents* Constraint

The condition which requires that "each pair of like-colored disks is connected via a chain of disks which travels horizontally or vertically through disks of the same color as the pair" takes a bit more work to express in the language of CSPs. Suppose we have the complete assignment set:

$$\alpha = \begin{Bmatrix} x_{1,1} \leftarrow v_{1,1}, x_{1,2} \leftarrow v_{1,2}, \dots, x_{1,n} \leftarrow v_{1,n} \\ x_{2,1} \leftarrow v_{2,1}, x_{2,2} \leftarrow v_{2,2}, \dots, x_{2,n} \leftarrow v_{2,n} \\ \vdots \\ x_{n,1} \leftarrow v_{n,1}, x_{n,2} \leftarrow v_{n,2}, \dots, x_{n,n} \leftarrow v_{n,n} \end{Bmatrix}$$

where $v_{i,j} \in \{\bullet, \circ\}$ for all $i, j \in \{1,2 \dots n\}$

so that α assigns a value to every variable in X (this corresponds to a grid in which every cell contains a disk). We can then then use α to define a graph $G(\alpha) = (X, E)$ which takes the entire variable set X as its vertex set. The edges of $G(\alpha)$ are constructed as follows:

$$E = \{ (x_{i,j}, x_{p,q}) \mid |i-p| + |j-q| = 1, \; v_{i,j} = v_{p,q} \}$$

so that two variables are connected by an edge if and only if the cells they correspond to are orthogonally adjacent and contain disks of the same color. We call $G(\alpha)$ the connectivity graph for α. The connectivity graph for a complete assignment set for a 3 by 3 Shirokuro grid is illustrated in Figure 5 below.

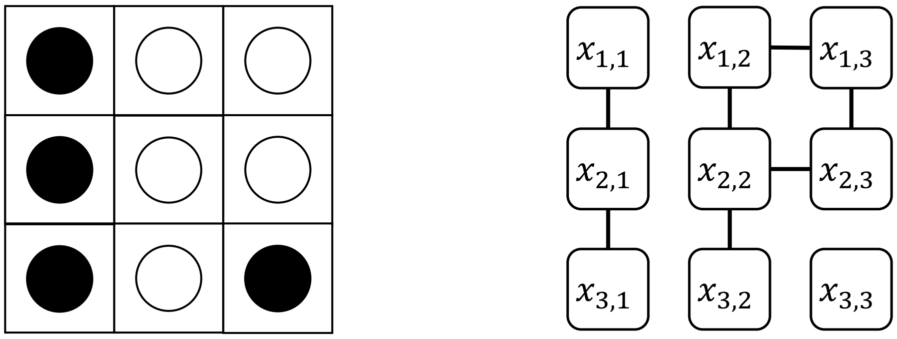


Figure 5: A complete assignment set (left) and its associated connectivity graph (right).

We can now define the following constraint:

$$TwoComponents = \left\{ \alpha \;\middle|\; \begin{array}{c} \alpha \text{ assigns a value to all variables in } X. \\ \text{The connectivity graph } G(\alpha) \\ \text{has 2 connected components} \end{array} \right\}$$

Thus $TwoComponents$ consists of all complete assignment sets in which every pair of like colored disks are connected by a chain of disks that travels horizontally or vertically through a sequence of disks that are of the same color as the pair.

2.5 The *Initial* Family of Constraints

A final set of constraints must be defined to account for the initial conditions of the Shirokuro puzzles. Recall that cells which initially contain a black or white disk are fixed and cannot be modified by the player. Thus we have subsets $X_\bullet \subset X$ and $X_\circ \subset X$ such that for every $x_\bullet \in X_\bullet$ we must define the constraint $Initial(x_\bullet) = \{\{x_\bullet \leftarrow \bullet\}\}$ and for every $x_\circ \in X_\circ$ we must define the constraint $Initial(x_\circ) = \{\{x_\circ \leftarrow \circ\}\}$.

2.6 Summary of the Shirokuro CSP Description

In summary, when we express Shirokuro as a CSP we get the pair (X, C) where the members of X corresponds to an individual cell and the members of C are constraints that restrict the combination of values that can be taken simultaneously by the variables in their scopes. Putting together the constraints defined above gives us our completed constraint set:

$$C = \{NoClump(i,j) \;\mid\; i,j \in \{1,2,3,\dots,n-1\}\} \cup \{TwoComponents\}$$
$$\cup \;\{Initial(x) \;\mid\; x \in X_\bullet \cup X_\circ\}$$

where the first term of the constraint set corresponds to the condition that no 2×2 area of the grid may contain 4 disks of the same color. The second term of the constraint set corresponds to the condition that pairs of like colored disks must be connected by a chain that travels vertically or horizontally through a sequence of disks that are the same color as the pair. The third term of the constraint set corresponds to the initial conditions of the Shirokuro puzzle.

CHAPTER 3

Ariadne's Thread and Backtracking Algorithms

The most straightforward way of solving a constraint satisfaction problem such as Shirokuro is to simply enumerate through all complete assignment sets until one is found that satisfies all of the constraints. This "brute force" method is computationally expensive, since the number of complete assignment sets for an $n \times n$ Shirokuro puzzle grows exponentially as 2^{n^2}. Fortunately, backtracking algorithms are frequently more efficient than brute force [2].

3.1 Ariadne's Thread

Backtracking algorithms are frequently introduced within the context of the ancient Greek legend of Theseus and the Minotaur. For examples, see [7], [8], [9], [10], and [11]. The protagonist of the epic story, depicted in the figure below, is a hero named Theseus.

Theseus sought to slay the humanoid bovine monstrosity known as the Minotaur. The Minotaur's lair was deep within a complicated labyrinth on the island of Crete. Fortunately for Theseus, his love interest Ariadne gave him a ball of thread to help him navigate through the maze. By gradually unwinding the thread as he walked, Theseus could distinguish areas of the maze which he had already visited from those that were yet to be explored.

Using Ariadne's thread, Theseus could have proceeded as follows to completely explore the maze:

Figure 6: Theseus in the Minotaur’s Labyrinth [12]

STEP 1: Pick any door which you have not walked through before and walk through it. Repeat step one until you can’t find any unexplored doors. Then proceed to step 2.

STEP 2: If you find yourself in a room where you have already walked through each door, proceed to the room you were in directly before visiting the current room for the first time. Continue from step 1.

A device which acts as a record of the available and exhausted options, here taking the form of Ariadne's thread, is the essence of the backtracking approach [13].

3.2 Backtracking Algorithms

Backtracking gains an advantage over brute force methods by exploiting a property of CSPs known as the *domino principle*: if an assignment set violates a given constraint, then all extensions (i.e. supersets) of that assignment set will also violate the constraint [14]. Brute force cannot exploit this property because only complete assignment sets are considered. The backtracking approach, on the other hand, iteratively grows assignment sets one assignment at a time until a complete solution is found. We can organize the set of all possible assignment sets into a tree structure where the *n*th level of the tree contains all of the assignment sets that assign values to the first *n* variables of a CSP's variable set. This is called the *variable assignment tree* of a CSP [5]. An example of a variable assignment tree is given in Figure 7. We can conceptualize the backtracking approach as a partial depth-first search of the variable assignment tree. While depth-first search exhaustively visits every node of a graph, backtracking algorithms seek to "prune" the variable assignment tree by detecting, as soon as possible, those partial assignment

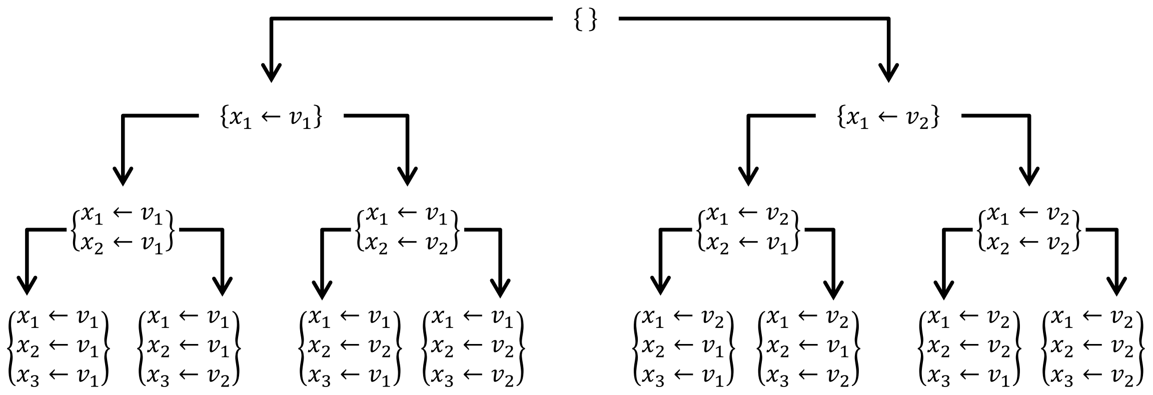

{}
$\{x_1 \leftarrow v_1\}$
$\{x_1 \leftarrow v_2\}$
$\{x_1 \leftarrow v_1, x_2 \leftarrow v_1\}$
$\{x_1 \leftarrow v_1, x_2 \leftarrow v_2\}$
$\{x_1 \leftarrow v_2, x_2 \leftarrow v_1\}$
$\{x_1 \leftarrow v_2, x_2 \leftarrow v_2\}$
$\{x_1 \leftarrow v_1, x_2 \leftarrow v_1, x_3 \leftarrow v_1\}$
$\{x_1 \leftarrow v_1, x_2 \leftarrow v_1, x_3 \leftarrow v_2\}$
$\{x_1 \leftarrow v_1, x_2 \leftarrow v_2, x_3 \leftarrow v_1\}$
$\{x_1 \leftarrow v_1, x_2 \leftarrow v_2, x_3 \leftarrow v_2\}$
$\{x_1 \leftarrow v_2, x_2 \leftarrow v_1, x_3 \leftarrow v_1\}$
$\{x_1 \leftarrow v_2, x_2 \leftarrow v_1, x_3 \leftarrow v_2\}$
$\{x_1 \leftarrow v_2, x_2 \leftarrow v_2, x_3 \leftarrow v_1\}$
$\{x_1 \leftarrow v_2, x_2 \leftarrow v_2, x_3 \leftarrow v_2\}$

Figure 7: The variable assignment tree for a CSP with variable set $\{x_1, x_2, x_3\}$, each variable has the domain $\{v_1, v_2\}$.

sets that cannot be extended to form a complete solution. Recall that such assignment sets are called *no-goods*. Since all CSPs fulfill the domino principle, we know that whenever a partial assignment set is found to violate some constraint, there is no need to inspect any extension of the violating assignment set. In terms of the variable assignment tree, the subtree rooted at the no-good is not explored, so that the graph traversed by a backtracking algorithm is potentially much smaller than the original variable assignment tree. This potentially smaller graph traversed by the backtracking algorithm is called the *backtracking search tree* [5]. A flow chart, Figure 8, illustrating a stack based backtracking process is illustrated on the following page.

3.3 Applying Backtracking to the Shirokuro Puzzle CSP

To solve Shirokuro puzzles with backtracking, we must invent ways to check if a given partial assignment set violates the constraints given in Section 2, where we gave a representation of the Shirokuro puzzle as a CSP with variable set X and constraint set C. Checking an assignment set against each of the constraints in the $NoClump$ family, which corresponds to the condition that no 2×2 region of the Shirokuro grid may contain 4 disks of the same color, is trivial.

Likewise, checking that an assignment set does not violate any of the $Initial$ family of constraints, which correspond to initial conditions, is trivial. These constraints are very easy to check because they all have very limited, local scopes. That is, to check that an assignment set does not violate a constraint in the $NoClump$ family, we only need concern ourselves with the 4 variables in the 2×2 region associated with the constraint. On the other hand, the scope of the $TwoComponents$ constraint is global in nature. The difficulties associated with the global nature of the $TwoComponents$ constraint are the topic of the next section.

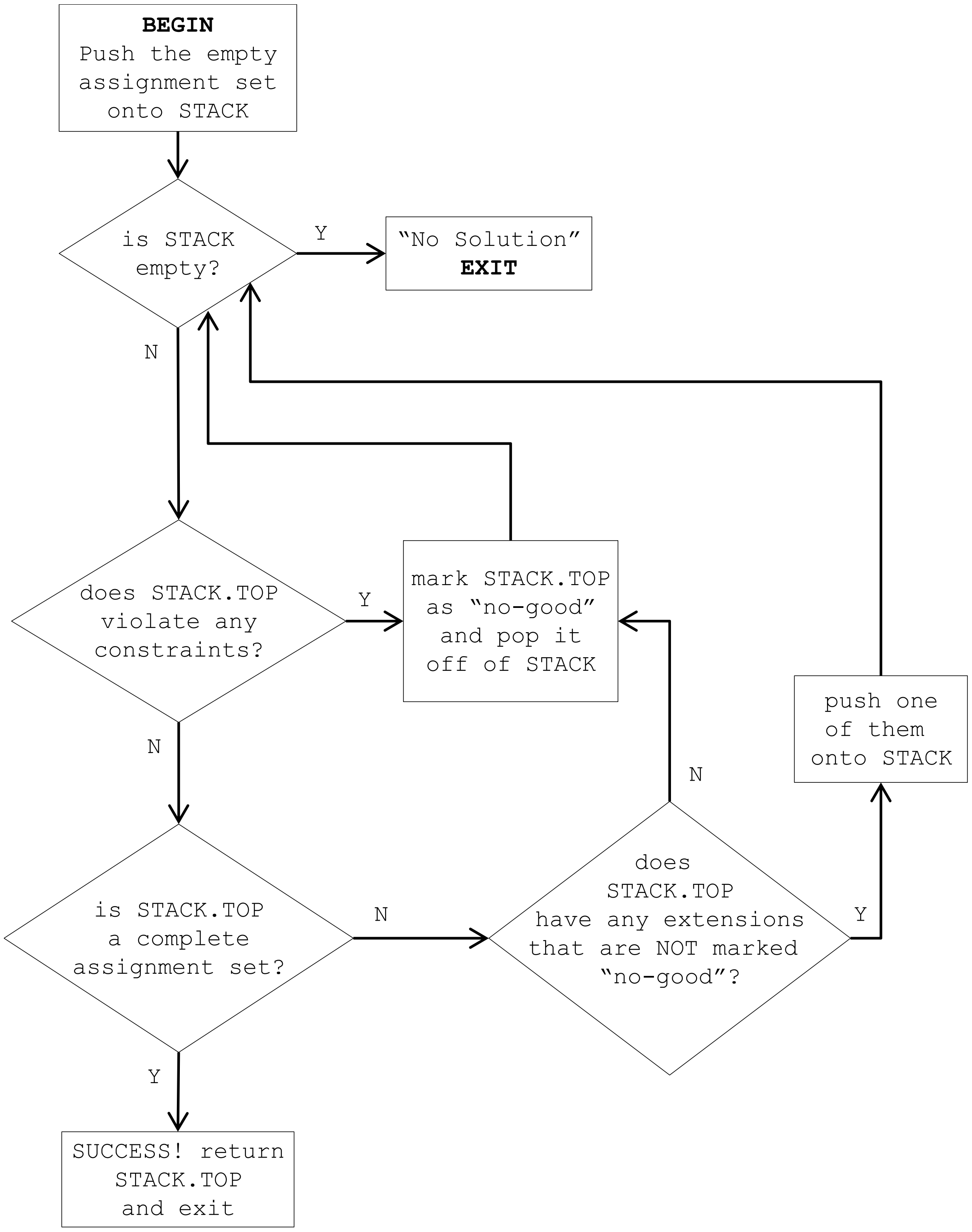

BEGIN
Push the empty assignment set onto STACK
is STACK empty?
Y
"No Solution" EXIT
N
does STACK.TOP violate any constraints?
Y
mark STACK.TOP as "no-good" and pop it off of STACK
N
is STACK.TOP a complete assignment set?
N
does STACK.TOP have any extensions that are NOT marked "no-good"?
N
Y
push one of them onto STACK
Y
SUCCESS! return STACK.TOP and exit

Figure 8: A flow chart for a stack based backtracking algorithm.

CHAPTER 4

Improving Performance with Relaxed Constraints

Shirokuro requires that each pair of like-colored disks be connected via a chain of disks which travels horizontally or vertically through disks of the same color as the pair. In chapter 2, we defined the $TwoComponents$ constraint, which consists only of those complete assignment sets whose associated connectivity graphs have only two components. Unfortunately, the scope of $TwoComponents$ is the entire variable set X. This means that the $TwoComponents$ constraint is only activated by complete assignment sets which assign a value to every variable. Since no partial assignment set can violate $TwoComponents$, a backtracking algorithm is not able to prune any part of the variable assignment tree on the basis of the $TwoComponents$ constraint. Fortunately, we can derive several additional constraints from $TwoComponents$ that are more local in scope and add them to our CSP representation of Shirokuro.

4.1 Relaxed Constraints

Our strategy is to invent a constraint c so that whenever a partial assignment set α violates c, any assignment set that is an extension of α which activates $TwoComponents$ also violates $TwoComponents$. We also require that any assignment set α which satisfies $TwoComponents$ also satisfies c. Such a constraint is called a *relaxation* of $TwoComponents$ [15]. Relaxed constraints represent logical consequences of the constraints that they relax and hence their use does not change the set of solutions to the CSP. However, relaxed constraints can have smaller scopes than the constraints they relax. For example, suppose we have the constraint $SumLessThanTen$ defined as:

$$SumLessThanTen = \left\{ \left\{ \begin{matrix} x_1 \leftarrow v_1,\ x_2 \leftarrow v_2,\ x_3 \leftarrow v_3, \\ x_4 \leftarrow v_4,\ x_5 \leftarrow v_5, \end{matrix} \right\} \middle| \sum_{i=1}^{5} v_i < 10 \right\}$$

where the scope of the constraint is $Scope(SumLessThanTen) = \{x_1, x_2, x_3, x_4, x_5\}$ and each x_i has the domain $\{1,2, \ldots ,9\}$. Then one potentially useful relaxation of $SumLessThanTen$ is the constraint

$$x1LessThan6 = \left\{ \{x_1 \leftarrow v_1\} \mid v_1 < 6 \right\}$$

Any assignment set which violates the $x1LessThan6$ will violate the $SumLessThanTen$ constraint because the minimum value of the sum $\sum_{i=1}^{5} v_i$ will be 6 + 1 + 1 + 1 + 1 = 10. Conversely, any assignment set which satisfies the $SumLessThanTen$ constraint will also satisfy the $x1LessThan6$ constraint, since the maximum value for x_1 would be 9 – 1 – 1 – 1 – 1 = 5. Thus, $x1LessThan6$ is a relaxation of $SumLessThanTen$; indeed like all relaxed constraints, it is a logical consequence of the constraint that it relaxes.

Because relaxed constraints have smaller scopes than the constraints they relax, they are easier to activate, and thus allow for the detection of no-goods at shallow depths of the variable assignment tree. This helps to prune the variable assignment tree, which results in a smaller backtracking search tree. We give several useful relaxations of the $TwoComponents$ constraint in this chapter.

4.2 The *NoX* Family of Constraints

Consider the partial assignment set represented by the illustration in Figure 9. The assignment set is characterized by a 2×2 region containing one pair of black disks and one pair of white disks. One pair of disks forms a Northwest-Southeast direction, while the other forms a Northeast-Southwest direction (left).

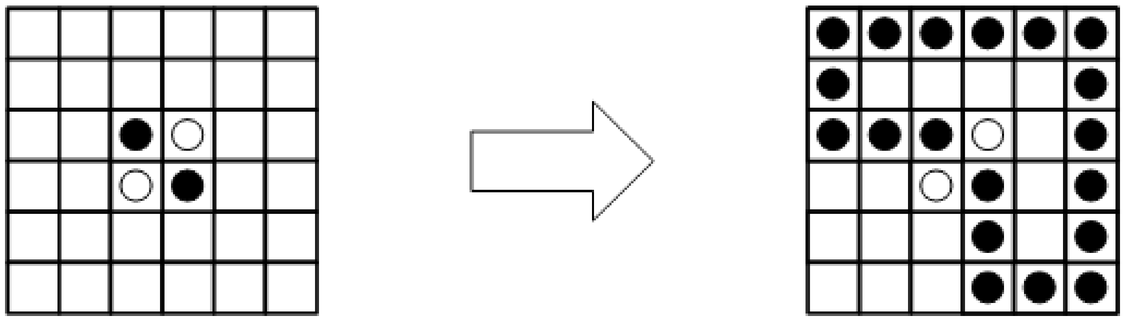


Figure 9: Assignment sets containing an “X” pattern (left) lead to the isolation of a pair of like-colored disks (right).

This configuration leads to the separation of a pair of disks (right). To see this, note that any connection that is made between the two black disks invariably forms a loop which completely encapsulates one of the white disks, making any connection between the two white disks impossible. Therefore for every element of $\{(i,j) \mid i,j \in \{1,2,3,\ldots,n-1\}\}$ we define the following constraint:

$$NoX(i,j) = \left\{ \left\{ \begin{matrix} x_{i,j} \leftarrow v_1,\ x_{i+1,j} \leftarrow v_2, \\ x_{i,j+1} \leftarrow v_3,\ x_{i+1,j+1} \leftarrow v_4 \end{matrix} \right\} \;\middle|\; \begin{matrix} v_k \in \{\bullet,\circ\} \text{ for } k \in \{1,2,3,4\}, \\ v_1 \neq v_4 \text{ OR } v_2 \neq v_3 \end{matrix} \right\}$$

So that $NoX(i,j)$ contains only those assignment sets that do not contain the problematic “X” configuration within the 2×2 region located at position (i,j). Note that each constraint in the NoX family of constraints is a relaxation of the $TwoComponents$ constraint, as desired.

4.3 The *BlackConnected* and *WhiteConnected* Constraint Families

Before going further, we need to define two additional graphs that can be associated with a given assignment set. Given an assignment set α, we generate a graph $G_\bullet(\alpha) = (V,E)$ which takes the following vertex and edge set. The edges of $G_\bullet(\alpha)$ are constructed as follows:

$$V = \{\, x \in X \ \mid (x \leftarrow \circ) \notin \alpha\}$$

$$E = \{ (x_{i,j}, x_{p,q}) \mid |i-p| + |j-q| = 1 \}$$

so that two variables are connected by an edge if and only if the cells they correspond to are orthogonally adjacent and do not contain white disks (i.e. they contain black disks or are empty). We call $G_\bullet(\alpha)$ the *black connectivity graph* for α.

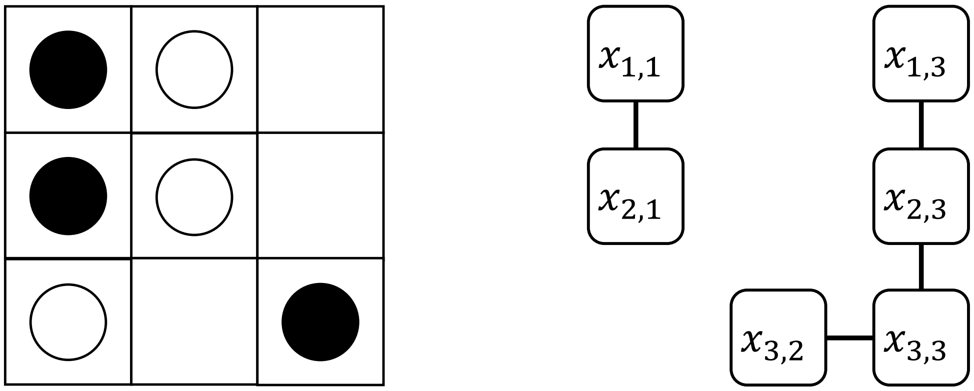


Figure 10: An incomplete assignment set (left) and its associated black connectivity graph (right). Compare with Figure 5, page 7.

Note that in Figure 10 above, $G_\bullet(\alpha)$ has two connected components. Since each component contains at least one black disk, there is at least one pair of black disks that cannot be connected to each other. Ergo, if the complete assignment set $\bar{\alpha}$ extends α, then $\bar{\alpha}$ will not satisfy the *TwoComponents* constraint, since the connectivity graph $G(\bar{\alpha})$ has at least 3 connected components. In general, any assignment set α is a no-good whenever there is a pair of black disks which is not found within the same connected component of the black connectivity graph $G_\bullet(\alpha)$. We can define a similar graph, the *white connectivity graph* $G_\circ(\alpha)$, for which similar results are found. We define the following two constraints for every subset X' of X as follows:

$$BlackConnected(X') = \left\{ \alpha \;\middle|\; \begin{array}{c} \alpha \text{ assigns a value to each variable in } X'. \\ \text{For every pair of black disks, there is} \\ \text{a path in } G_\bullet(\alpha) \text{ which connects them.} \end{array} \right\}$$

$$WhiteConnected(X') = \left\{ \alpha \;\middle|\; \begin{array}{c} \alpha \text{ assigns a value to each variable in } X'. \\ \text{For every pair of white disks, there is} \\ \text{a path in } G_{\circ}(\alpha) \text{ which connects them.} \end{array} \right\}$$

Note that since these constraints are defined for every subset of X, we can use them to identify no-goods at any level of the backtracking search tree.

4.4 The BlackEdgeConnected Family of Constraints

Our final set of constraints is obtained by exploiting an interesting property of the boundary of the Shirokuro grid. Consider the partial assignment set illustrated in Figure 11 (left), in which 4 disks of alternating colors (●,○,●,○) are found along the edge of the grid. Note that any connection that is drawn between the two black disks would necessarily bisect the Shirokuro grid, thereby isolating the pair of white discs from one another, as in Figure 11 (right). Thus, if two black disks lie along the boundary of the grid, then they must be connected by a chain of black disks and/or empty cells, all of which also lie along the boundary.

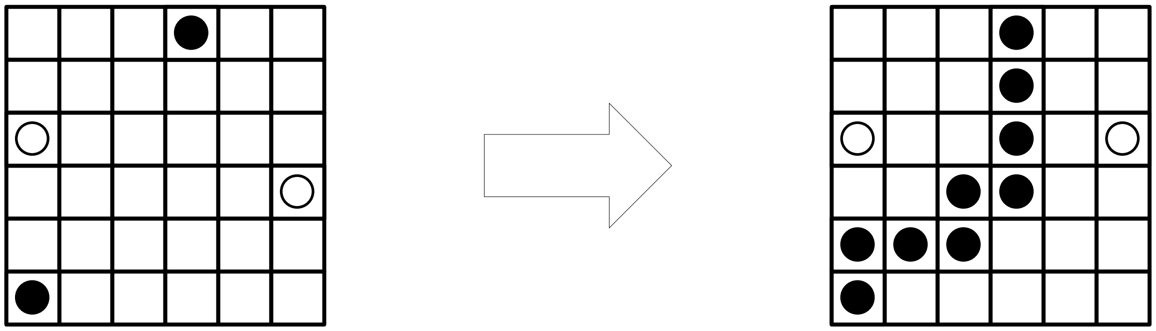


Figure 11: Assignment sets containing 4 disks of alternating colors along edge of the grid (left) lead to the isolation of a pair of like-colored of disks (right).

To write this in the language of CSPs requires that we define yet another connectivity graph. For any assignment set α let the *black edge connectivity graph* $G_{\bullet}^{e}(\alpha) = (V, E)$ have the following vertex and edge sets:

$$V = \left\{ x_{i,j} \in X \;\middle|\; i,j \in \{1,n\},\quad (x_{i,j} \leftarrow \circ) \notin \alpha \right\}$$

$$E = \left\{ \left(x_{i,j}, x_{p,q}\right) \;\middle|\; |i-p| + |j-q| = 1 \right\}$$

Note that the black edge connectivity graph $G_{\bullet}^{e}(\alpha)$ is a vertex-induced subgraph of the previously defined black connectivity graph $G_{\bullet}(\alpha)$, with the vertices restricted to those corresponding to cells that lie on the edge of the Shirokuro grid.

If $X^{e} = \left\{ x_{i,j} \in X \;\middle|\; i,j \in \{1,n\} \right\}$, then for every subset $X^{e\prime} \subset X^{e}$ we define the following constraint:

$$BlackEdgeConnected(X^{e\prime}) = \left\{ \alpha \;\middle|\; \begin{matrix} \alpha \text{ assigns a value to each variable in } X^{e'}, \\ G_{\bullet}^{e}(\alpha) \text{ is connected.} \end{matrix} \right\}$$

After some thought, it should become clear that it is not necessary to define a similar family of constraints based on white disks; this constraint would be redundant in this case.

CHAPTER 5

Further Improvements

The preceding text gives a basic theoretical foundation to those who wish to represent the Shirokuro puzzle as a CSP and to solve the CSP using rudimentary backtracking techniques. In this chapter, we discuss several avenues of improvement which may enhance the performance of the Shirokuro backtracking algorithm.

5.1 Thrashing

While the backtracking approach is often times more efficient than simple brute force methods, backtracking is vulnerable to a pathological phenomenon known as thrashing [2]. Freuder and Mackworth define thrashing as "the repeated exploration of failing subtrees of the backtracking search tree that are essentially identical–differing only in assignments to variables irrelevant to the failure of the subtree [16]." Consider the assignment set depicted in the following figure:

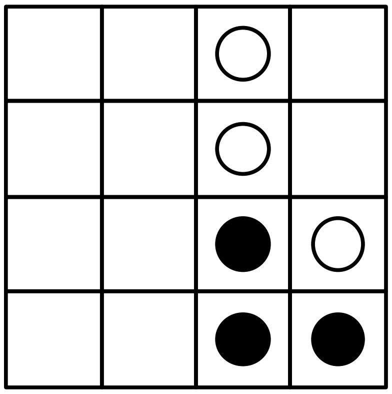


Figure 12: An assignment set which cannot be extended to a valid solution

It is easy for human eyes to see that any attempt to extend this partial assignment set to form a valid solution will end in failure: the two empty cells in the upper right corner of the grid have become isolated from the black disks in the lower right corner. Therefore none of these empty cells may be assigned a black disk. However, assigning a white disk to both of those cells results in an assignment set which violates a member of the *NoClump* family of constraints.

Despite the simplicity of this argument, the simple backtracking algorithm could visit 70 or more extensions of this assignment sets before finally marking it as a no-good. An example of one of these large backtracking search trees, rooted at the above assignment set, is illustrated in Figure 13, below. This is an example of thrashing; exploring various extensions which assign values to the two leftmost columns of cells does nothing to address the ultimate cause of the failures which eventually occur.

Thrashing is serious business: The amount of thrashing that takes place is often the greatest factor affecting the runtime of a backtracking algorithm [16]. In section 5.2 and 5.3 we briefly discuss some of the approaches that can be used to reduce thrashing behavior.

5.2 Variable Ordering

In Figure 8, we gave the flow chart for a simple stack based backtracking algorithm. The reader should note that the algorithm does not specify the order in which variables should be added to the scope of the assignment set that is being constructed. We have extracted the relevant portion of the flow chart below for the reader's reference.

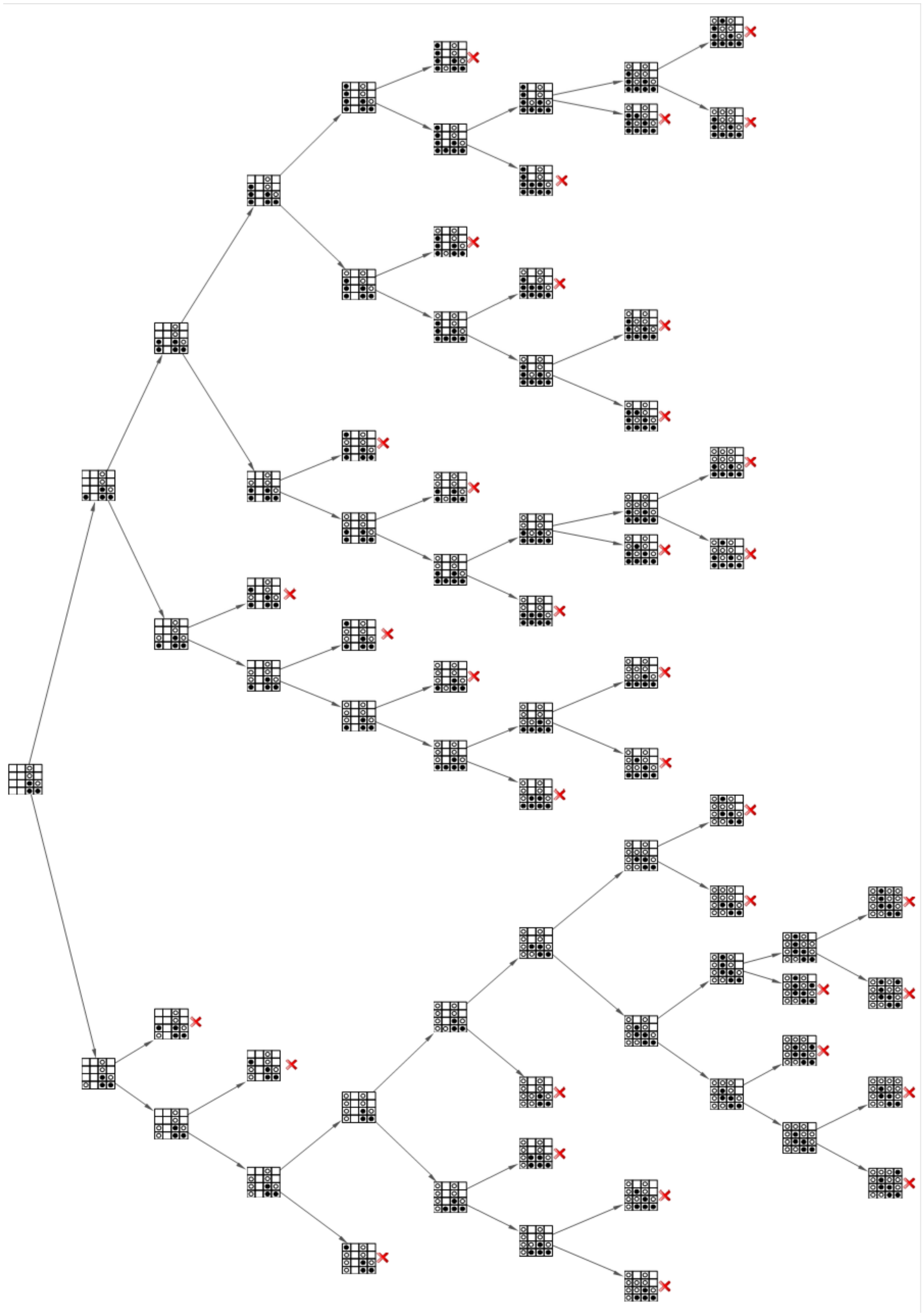


Figure 13: An example of thrashing in the simple backtracking algorithm

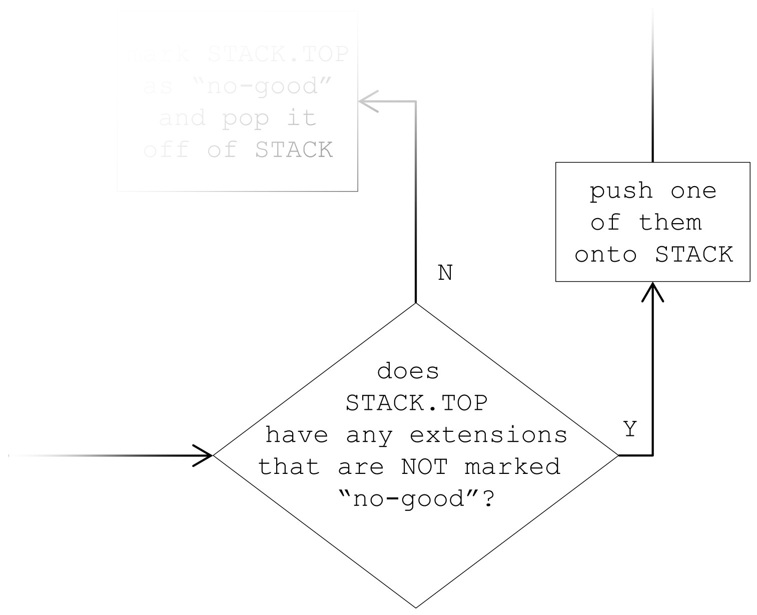

mark STACK.TOP as "no-good" and pop it off of STACK
push one of them onto STACK
N
does STACK.TOP have any extensions that are NOT marked "no-good"?
Y

Figure 14: The portion of the backtracking flow chart which shows that the order of variables is not specified in simple backtracking algorithms. See Figure 8, page 13.

Although simple backtracking makes variable assignments in arbitrary order, trashing behavior can sometimes be dramatically reduced by using a more careful variable ordering [17]. The backtracking search tree illustrated below is rooted at the same assignment set as the large thrashing example in Figure 13. However, while the backtracking search tree in Figure 13 contains 71 partial assignment sets, the backtracking search tree illustrated below contains only 5 partial assignment sets. This reduction in size of the backtracking search tree is a direct result of the careful variable ordering used in the construction of the smaller tree; as a result of this ordering, none of the variables corresponding to the two left-most columns of cells, which are irrelevant to the ultimate cause of the subtree's failure, are ever assigned any values.

Advanced backtracking algorithms make use of a powerful technique called dynamic variable ordering (DVO). In DVO, the order in which variable assignments are

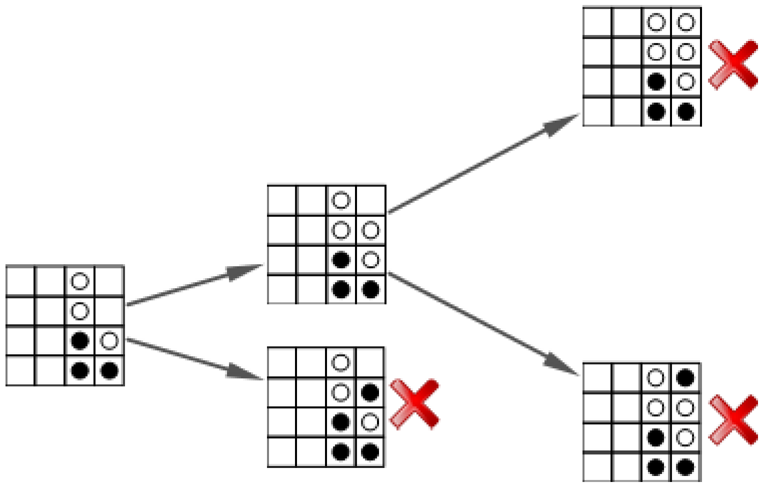


Figure 15: Carefully ordering variable assignments can dramatically reduce thrashing. Compare to Figure 13, page 23.

made is calculated during the search itself, so that each branch of the backtracking search tree can potentially have a different variable ordering. The general strategy is to assign values to variables that are likely to result in failure as soon as possible in order to maximize pruning of the variable assignment tree. The details of how this strategy can be implemented are beyond the scope of this thesis. However, because several of the Shirokuro constraints are defined in terms of graph connectivity, we believe that identifying the articulation points (nodes of a connected graph that disconnect the graph when they are removed) of a graph will play a crucial role in the formulation of effective DVO approaches to Shirokuro. In Figure 16, we provide an example in which the variable associated with the articulation point of the black connectivity graph (marked with a red star) is the first to undergo assignment. Note that this results in the immediate discovery of a no-good, as desired. Articulation points can be discovered via a clever linear time algorithm based on spanning trees and depth first search. See [18] for a thorough treatment of the subject. See also [19] and [20] for a more accessible

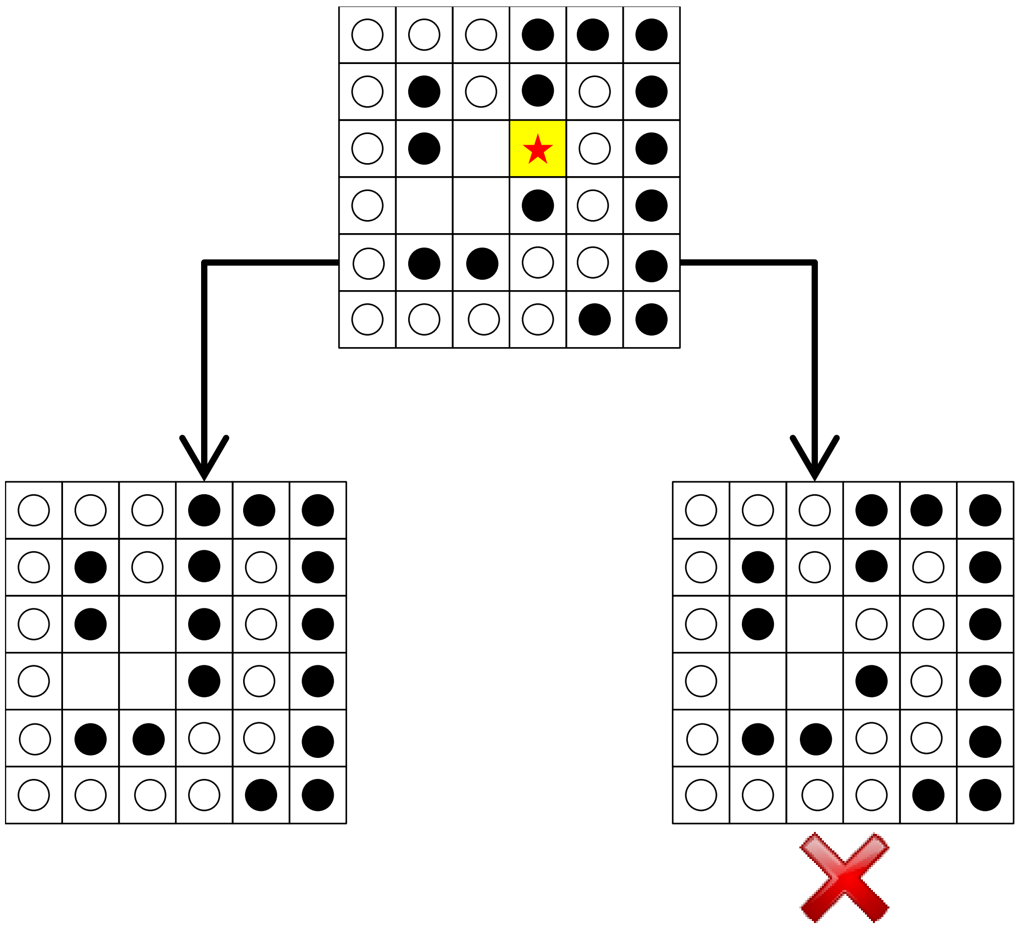


Figure 16: Articulation points in dynamic variable ordering

explanation. Information on separation pairs, which may also be relevant, can be found in [21].

5.3 Forward Checking

Forward checking (FC) is a simple constraint propagation [22] technique which can further reduce thrashing behavior. The idea behind FC is simple to explain. Suppose ouour backtracking algorithm is about to extend the assignment set α. Before doing so, we make a list of all the unassigned variables, i.e. the set of variables that are not in the scope of α, i.e. the compliment of $Scope(\alpha)$, which we denote $\overline{Scope(\alpha)}$. Now, for each variable $x \in \overline{Scope(\alpha)}$, we examine the set $Dom(x)$ and temporarily remove from it all values that would result in a constraint violation if used to extend the assignment set α (these domain values are restored if and when the backtracking algorithm discovers that α is a no-good). Of course, each of variables in the Shirokuro CSP has the domain {●,○}, so that removing a value from the domain of a Shirokuro variable is equivalent to assigning a value to that variable. Therefore Shirokuro FC causes several assignments to be made in rapid succession: the first assignment in each group of assignments is exploratory; this first assignment is then followed by a series of supplementary assignments that are "forced" into existence by the first assignment.

The figure below contains a few examples of how forward checking would work in a Shirokuro backtracking algorithm. For clarity, in each example we evaluate forward checking on the basis of only one family of constraints.

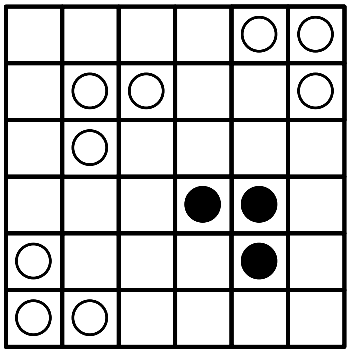
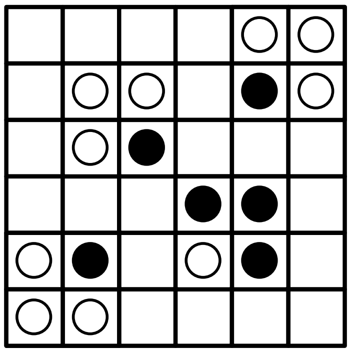
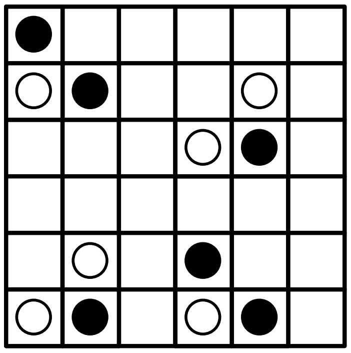
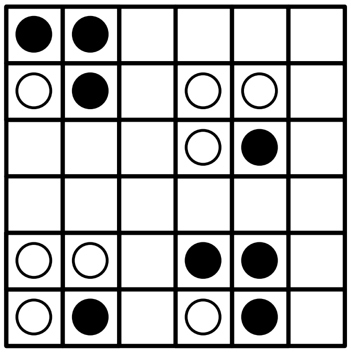
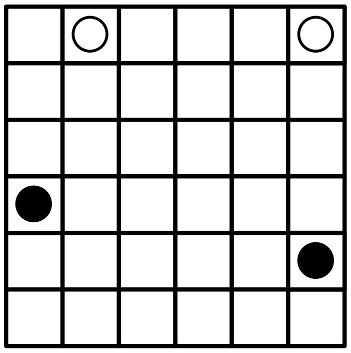
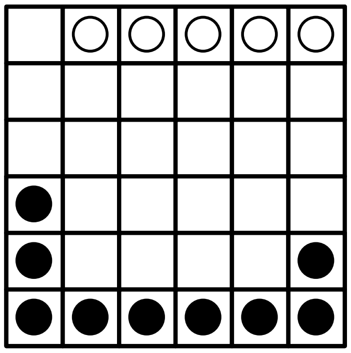
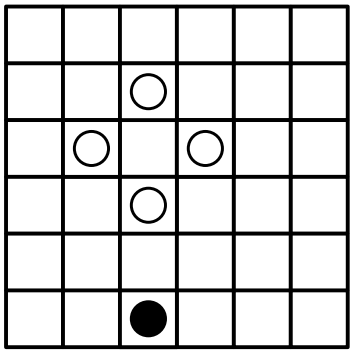
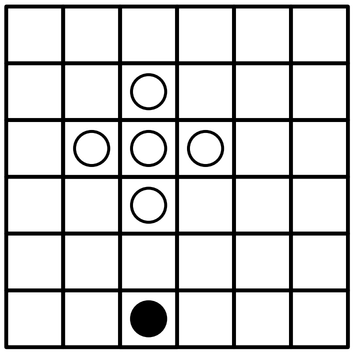
constraint family basis for FC	assignment set before FC	assignment set after FC
$NoClump$		
NoX		
$BlackEdgeConnected$		
$BlackConnected$		

Figure 17: The effects of forward checking on the basis of various constraint families

CHAPTER 6

Future Work

In the following sections we present several lines of inquiry which were superficially explored but not conclusively investigated. We document these explorations here so that they may serve as the basis for further investigations, both by the author and by the readers of this thesis.

6.1 Spanning Tree Maintenance

Verifying the constraints within the $BlackConnected$ and $WhiteConnected$ families require that graphs be constructed so that their connected components can be counted. Moreover, the Dynamic Variable Ordering approach we discussed above requires that articulation points be identified at every step of the backtracking process. These tasks rely on using depth first search to construct spanning trees which span the black and white connectivity graphs. Although Tarjan has shown that the building of spanning trees and the identification of articulation points can be achieved in linear time [18], reconstructing these spanning trees from scratch at every step of backtracking process might include many redundant steps. It would be more desirable to use dynamic algorithms which maintain, rather than rebuild, their data structures, so that a small change in graph structure would correspond to a small "updating" computation. Dynamic algorithms which allow for vertex and edge insertions are described in [23]. However, we require an algorithm which allows for vertex and edge deletions as well. A large number of references to dynamic graph algorithms can be found in [21]. An appropriate dynamic algorithm for Shirokuro may well lie among them.

6.2 Solving The Shirokuro CSP With Evolutionary Algorithms

One approach to solving Constraint Satisfaction Problems is to recast the CSP as an optimization problem that can be solved through the use of biologically inspired evolutionary algorithms (EAs). A brief description of a typical EA is given here. A more extensive explanation of EAs is given [24].

The classical optimization methods typically taught in graduate level courses on mathematical optimization are relatively efficient at solving “linear, quadratic, strongly convex, unimodal, and separable” problems [25]. EAs, on the other hand, were developed with robustness (broad applicability) in mind. They are often used to optimize functions that are “discontinuous, nondifferentiable, multimodal, noisy, and otherwise unconventional” [25].

EAs were inspired by the process of Darwinian natural selection observed in the biological world. According to Darwinism, populations of organisms adapt to the environment through the mechanisms of selection, mutation, sexual recombination and sexual reproduction. In EAs, a simulation of this process is set up so that a population of candidate solutions (also called “individuals”) adapt to a virtual environment in which natural selection is replaced by an artificial selection operator. The latter makes use of fitness data to probabilistically eliminate the worst individuals. New individuals are then generated from the surviving individuals by means of the mutation operators and/or recombination operators. An example flow chart for a simple evolutionary algorithm is illustrated below:

For the purpose of Shirokuro, the fitness function might reflect, for example, the number of constraints that are violated by any given individual, so that individuals which

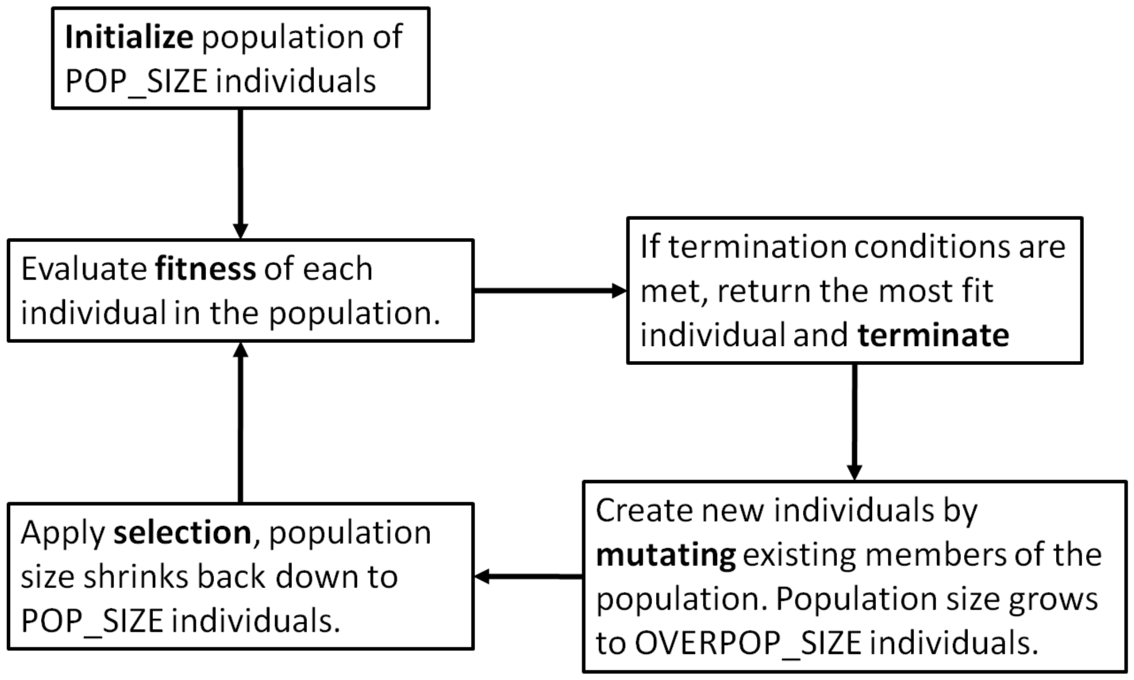

Initialize population of POP_SIZE individuals
Evaluate **fitness** of each individual in the population.
If termination conditions are met, return the most fit individual and **terminate**
Create new individuals by **mutating** existing members of the population. Population size grows to OVERPOP_SIZE individuals.
Apply **selection**, population size shrinks back down to POP_SIZE individuals.

Figure 18: Flow chart for a typical evolutionary algorithm.

violate fewer constraints are given a reproductive advantage. The problem with this approach is that much information is lost when all of the information regarding constraint violations is aggregated into a single number. Another problem is that the traditional mutation and recombination operators found in most EA operators are “blind” to constraints, meaning that there is no guarantee that the offspring will satisfy the same constraints as its parents. Techniques for dealing with these issues are given in [26].

6.3 The Computational Complexity of Shirokuro

Understanding the complexity of the computational problem one is attempting to solve is important because such an understanding allows one to form reasonable goals and expectations regarding the efficiency of any algorithm designed to solve the problem [27]. While the basic concepts of computational complexity theory are beyond the scope of this thesis, an accessible introduction to the topic is provided in [28]. A more comprehensive treatment can be found in [27].

To express Shirokuro as a decision problem, we can ask the question “does a given instance of Shirokuro have a solution?” It is relatively trivial to show that the Shirokuro decision problem lies in NP (Non-deterministic Polynomial time). All that is needed is a polynomial time algorithm that can verify a given candidate solution. Since depth first search runs in linear time, this is a simple task indeed.

Other than the fact that Shirokuro is a member of NP, we know nothing of the computational complexity of Shirokuro. While proofs exist that related puzzles, such as Sudoku, are NPC (NP-Complete) [29], such a proof that Shirokuro is NPC has eluded us. Assuming that P (Polynomial time) is not equal to NP, the possibility exists that Shirokuro exists neither in P nor in NPC, but instead lies in NPI (NP-Intermediate, see [30]). NPI is a sad place for a decision problem to find itself; proving that a problem lies in NPI would indirectly prove that P is not equal to NP. Thus proving that Shirokuro lies in NPI is at least as hard as proving that that P is not equal to NP, which is seen by many as the most important problem in computer science; indeed we may never know for sure. Thus if Shirokuro lies in NPI then its complexity might remain unknown forever. Nevertheless, it is the opinion of the author that the complexity of Shirokuro is worth researching. However, we must warn against working on this particular problem under any kind of deadline; if Shirokuro is in NPI, then even a lifetime may not provide enough time.

CHAPTER 7

Conclusion

In this thesis we provided a theoretical foundation for solving Shirokuro puzzles using backtracking techniques. This included a description of Shirokuro as a constraint satisfaction problem (CSP), a description of backtracking algorithms, and an overview of several performance enhancing techniques, such as relaxed constraints, variable ordering and forward checking. Finally we outline several potential avenues of future research, some of which could lead to more efficient Shirokuro backtracking algorithms, and some of which are unrelated to the backtracking approach.

References

[1] Helmut Simonis, "Sudoku as a constraint problem," in *CP Workshop on Modeling and Reformulating*, Sitges (Barcelona), Spain, 2005, pp. 13-27.

[2] Vipin Kumar, "Algorithms for Constraint-Satisfaction Problems: A Survey," *AI Magazine*, vol. 13, no. 1, pp. 32-44, 1992.

[3] Edward Tsang, "A Glimpse of Constraint Satisfaction," *Artificial Intelligence Review*, vol. 13, no. 3, pp. 215-227, June 1999.

[4] Donald E Knuth, "Estimating the Efficiency of Backtrack Programs," *Mathematics of Computation*, vol. 29, no. 129, pp. 121-136, January 1975.

[5] Fahiem Bacchus. A Uniform View of Backtracking. [Online]. http://www.cs.toronto.edu/~fbacchus/Papers/uniform-backtracking.pdf

[6] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*.: Prentice Hall, 2010.

[7] Djordje M Kadijevic, "Backtracking in Basic: How to Escape from the Labyinth," *Teaching Mathematics and its Applications*, vol. 10, no. 2, pp. 64-73, 1991.

[8] Loren P Meissner and Elliott Irving Organick, *Fortran 77: Featuring Structured Programming*, 3rd ed.: Addison-Wesley, 1980.

[9] Eric S Roberts, *Programming Abstractions in C: A Second Course in Computer Science*.: Addison Wesley, 1997.

[10] Philip Nicholas Johnson-Laird, *The Computer and the Mind: An Introduction to Cognitive Science*.: Harvard University Press, 1989.

[11] Micheal Mepham. (2005) Sudoku.org.uk. [Online]. http://www.sudoku.org.uk/PDF/solving_sudoku.pdf

[12] Sir Edward Burne-Jones. Pre-Raphaelite Online Resource. [Online]. http://www.preraphaelites.org/the-collection/1927p594/tile-design-theseus-and-the-minotaur-in-the-labyrinth/

[13] Wikipedia contributors. Wikipedia, The Free Encyclopedia. [Online]. http://en.wikipedia.org/wiki/Ariadne's_thread_(logic)

[14] Gabriel Valiente, *Algorithms on Trees and Graphs*. New York: Springer, 2002.

[15] Willem-Jan van Hoeve and Irit Katriel, "Global Constraints," in *Handbook of Constraint Programming*, Francesca Rossi, Peter van Beek, and Toby Walsh, Eds. New York: Elsevier, 2006, pp. 169-208.

[16] Eugene C Freuder and Alan K Mackworth, "Constraint Satisfaction: An Emerging Paradigm," in *Handbook of Constraint Programming*, Francesca Rossi, Peter van Beek, and Toby Walsh, Eds.: Elsevier, 2006, ch. 2, pp. 13-23.

[17] Fahiem Bacchus and Paul van Run, "Dynamic Variable Ordering in CSPs," in *Principles and Practice of Constraints Programming (CP-95)*, Cassis, France, 1995, pp. 258-275.

[18] Robert Endre Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 146-160, 1972.

[19] Krishnaiyan Thulasiraman and M. N. S. Swamy, *Graphs: Theory and Algorithms*.: John Wiley and Sons, 1992.

[20] Mark Allen Weiss, *Data Structures and Algorithm Analysis in C*.: Addison-Wesley, 1997.

[21] F Kammer and H Taubig, "Connectivity," in *Network Analysis: Methodological Foundations*, Ulrik Brandes and Thomas Erlebach, Eds.: Springer, 2005, ch. 7, pp. 143-177.

[22] Christian Bessiere, "Constraint Propagation," in *Handbook of Constraint Programming*, Francesca Rossi, Peter van Beek, and Toby Walsh, Eds., 2006, ch. 3, pp. 29-84.

[23] Jeffery Westbrook and Robert E Tarjan, "Maintaining Bridge-Connected and Biconnected Components On-Line," *Algorithmica*, vol. 7, pp. 433-464, 1992.

[24] David B Fogel, *Evolutionary computation: toward a new philosophy of machine intelligence*.: John Wiley and Sons, 2006.

[25] Hans-Paul Schwefel, "Anvantages (and disadvantages) of evolutionary computation over other approaches," in *Evolutionary Computation: Basic algorithms and operators*, David B Fogel, Thomas Bäck, and Zbigniew Michalewicz, Eds. New

York: Taylor and Francis Group, LLC, 2000, ch. 3, pp. 20-22.

[26] A E Eiben, "Evolutionary Algorithms and Constraint Satisfaction: Definitions, Survey, Methodology and Research Directions," in *Theoretical Aspects of Evolutionary Computing*, Leila Kallel, Bart Naudts, and Alex Rogers, Eds.: Springer, 2001, pp. 13-30.

[27] Michael R Garey and David S Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*.: W. H. Freeman and Company, 1979.

[28] Michael Sipser, *Introduction to the Theory of Computation*.: Thomson Course Technology, 2006.

[29] G Kendall, A Parkes, and K Spoerer, "A Survey of NP-Complete Puzzles," *ICGA Journal*, vol. 31, no. 1, pp. 13-34, 2008.

[30] Richard E Ladner, "On the Structure of Polynomial Time Reducibility," *Journal of the ACM*, vol. 22, no. 1, January 1976.

[31] Ian P Gent, Karen E Petrie, and Jean-François Puget, "Symmetry in Constraint Programming," in *Handbook of Constraint Programming*, Francesca Rossi, Peter van Beek, and Toby Walsh, Eds. New York: Elsevier, 2006, pp. 329-376.

APPENDIX:

Internet Resources

A worksheet from Spirit Magazine containing 4 easy Shirokuro puzzles:

http://cs.calstatela.edu/wiki/images/4/46/ShirokuroSpirit.pdf

A poster on Shirokuro by Benjamin James Bush:

http://cs.calstatela.edu/wiki/images/d/de/Shirokuro.pdf

A poster on Shirokuro by Alexandre Lomovtsev:

http://cs.calstatela.edu/wiki/images/c/ce/Shirokuro_Solver.pdf

Shirokuro Solver Project page by Alexandre Lomovtsev:

http://cs.calstatela.edu/wiki/index.php/Courses/CS_491ab/Winter_2009/
Alexandre_Lomovtsev

A Python package for constraint satisfaction:

http://labix.org/python-constraint

A Python package for networks and graphs:

http://networkx.lanl.gov/

A Python package for evolutionary algorithms:

http://pyevolve.sourceforge.net/

Josh Buhler’s Shirokuro Puzzle App for Windows Phone 7:

http://fogodev.com/post/3184856247/shirokuro-is-live