Internal Report 2012–01 June 2012

Universiteit Leiden
Opleiding Informatica

Playing Games

The complexity of Klondike, Mahjong,

Nonograms and Animal Chess

Jan N. van Rijn

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

We analyse various aspects of the Constraint Logic framework, from the book "Games, Puzzles and Computation" by Hearn and Demaine, and we present some complexity results on specific games.

We report on the half-crossover and the crossover gadget. Amongst other results, we have found out that these constructions lack some features that are needed to make a reduction from Bounded Nondeterministic Constraint Logic to Planar Bounded Nondeterministic Constraint Logic. Planar Bounded Nondeterministic Constraint Logic is NP-complete nonetheless, hence we can use it to reduce other problems from.

We present a proof that both Klondike and Mahjong are NP-complete, by reduction from Bounded Nondeterministic Constraint Logic. We show that determining whether a Nonogram has a solution and finding a solution is NP-complete, by reduction from Planar Bounded Nondeterministic Constraint Logic. Finally, we demonstrate a proof that Dou Shou Qi, also known as Animal Chess, is PSPACE-hard, by reduction from Bounded Two-Player Constraint Logic.

Contents

List of Figures

Chapter 1

Introduction

The field of Computational Complexity focuses on classifying computational problems according to their inherent difficulty, and relating those classes to each other. By expressing the maximum number of recourses (time and memory) needed in order to solve a specific problem as a function of the input, we can classify problems in so-called complexity classes. Some well-known complexity classes are P, NP, PSPACE and EXPTIME.

For each complexity class X, a problem is called X-*hard* if solving it is at least as hard as the hardest problem in X. Polynomial factors in the difficulty are ignored. A problem is said to be in X if we can solve the problem using at most the number of recourses specified for complexity class X. If a problem is both in X and a problem is X-*hard*, we classify the problem as X-*complete*.

Our goal is to analyze the computational complexity of games and puzzles. For each game (or puzzle), we define a computational problem or question relevant for that game, and determine what the complexity class of this problem is. A typical question would be whether the player can win the game, from a given situation. We stick to the convention of saying that the game is in the complexity class, rather than the self-defined problem associated to it.

Usually it is not hard to determine in which complexity class a certain problem is. In order to do so, one should compose an algorithm that uses at most the number of recourses specified for this complexity class. Showing that a problem is amongst the hardest problems in a certain complexity class however is not easy. One technique of doing so is using a *reduction*. By taking a problem for which it is already proven that it is X-*complete* (B) and showing that solving the problem in which we are interested (A) is at

least as hard as solving B, we can conclude that A is *X-hard*. We usually do this by showing a way to transform problem B into problem A. We then say $B \leq_p A$.

In "Games, Puzzles and Computation", by Hearn and Demaine, a framework containing different so-called graph games specific for every major complexity class is created. These graph games are created in order to reduce to other games. Reducing a game from one of these graph games appears to be quite easy. For example, in order to reduce from Nondeterministic Constraint Logic, which is a PSPACE-complete puzzle introduced in [6], one only has to construct two gadgets which simulate AND and OR gate behaviour.

We will examine this framework and use it to classify other games and puzzles in their respective complexity classes. Most of the puzzles we classify are already classified by other researches. We consider our contribution on these problems to use a reduction from a problem in the framework of [6]. In many cases the proof is highly visual and easy to understand.

This thesis can be divided in two parts. In the first chapters we will analyse the constraint logic framework itself, and present some theoretical work. From Chapter 5 on we will present some complexity results, obtained by using the constraint logic framework. In Chapter 2 we will give a formal definition of Nondeterministic Constraint Logic and Bounded Nondeterministic Constraint Logic, which are the puzzles defined by [6], and note on their planarity. In Chapter 3 and Chapter 4 we take a look at the Bounded NCL half-crossover and crossover gadget, constructed by the authors of [6], and analyse what properties these gadgets have. From Chapter 5 and on, we present the complexity results we have obtained during this research. In Chapter 5 we will show that Klondike is NP-complete. In Chapter 6 we will show that Mahjong Solitaire is NP-complete. In Chapter 7 we will show that Nonograms are NP-complete. In Chapter 8 we will show that Animal Chess is PSPACE-hard. Chapter 9 contains our initial results on the complexity of Rummikub, however this problem remains open. Finally, in Chapter 10 our conclusions are presented and some proposals for future work are made.

This master thesis is written as a partial fulfillment of the requirements of the degree Master of Science in the Leiden Institute of Advanced Computer Science (LIACS) of Leiden University, and is supervised by Hendrik Jan Hoogeboom and Walter Kosters.

Chapter 2

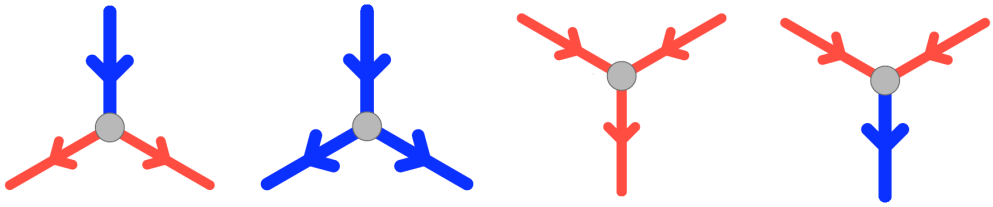
Nondeterministic Constraint Logic

In this chapter we will give a formal definition of *Nondeterministic Constraint Logic* and *Bounded Nondeterministic Constraint Logic*, as introduced by Hearn and Demaine [6]. We will also introduce some definitions and discuss their planarity.

2.1 Definition

Nondeterministic Constraint Logic (*NCL*), as presented in [6], is a *graph game* which is played on a *constraint graph* $CG = (V, E)$ with vertex set V and edge set E. A constraint graph is a weighted directed graph, where each edge has a weight in $\{1, 2\}$. Each vertex has a maximal in/outdegree of 3. The *inflow* of a vertex is defined to be the sum of all weights of the edges that are directed inward. Each vertex has a nonnegative *minimum inflow*. Vertices adjacent to three edges are drawn big and have a minimal inflow of 2, vertices adjacent to only two edges are drawn small and have a minimal inflow of 1. A *configuration* is a constraint graph where for each edge it is specified to which vertex it is pointing. A configuration of a constraint graph is legal if and only if for each vertex it holds that the inflow is equal to or larger than its minimum inflow. A *state* is defined to be a legal configuration. A move of the player consists of the reversal of a single edge, such that it results in a legal configuration. The goal of the game is to reverse a certain *target edge*, after a series of moves. The restricted form of this game, in which each

edge can only be reversed once at most, is called Bounded Nondeterministic Constraint Logic (*Bounded NCL*). Since every edge can only be flipped once, not only the direction of the edge is important for the description of a state, but also the Boolean property whether it has already been flipped or not. A state where none of the edges has been reversed is called an *initial state*. In [6] it is proven that Bounded NCL is NP-complete, even when the constraint graph consists of only the gadgets shown in Figure 2.1. In this figure an edge is drawn blue (and thick) if and only if it has a weight of 2; an edge is drawn red (and thin) if and only if it has a weight of 1. We will stick to this convention for the rest of this report.



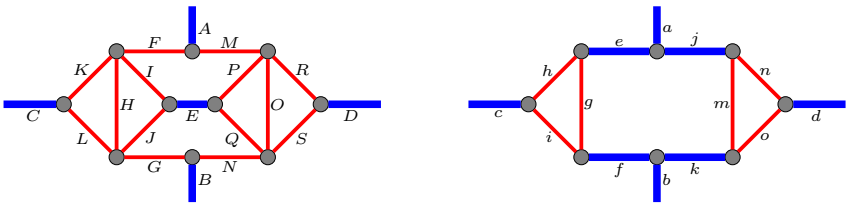
(a) AND (b) OR (c) CHOICE (d) FANOUT

Figure 2.1: Bounded NCL vertices.

2.2 Planar Bounded NCL

In [6] it is shown that every Bounded NCL Graph has an equivalent planar Bounded NCL Graph. Every pair of crossing edges can be replaced by the gadget shown in Figure 2.2(a), which consists of the vertices shown in Figure 2.1 and an additional vertex type, the red-red-red-red vertex. The authors of [6] show that this vertex type can be simulated using merely the AND-vertex and the OR-vertex, as shown in Figure 2.2(b).

The edges of these gadgets can be divided into two disjoint sets. The *internal edges* are all edges which are adjacent to two vertices that are both element of the same gadget. The *external edges* are the edges which are adjacent to only one vertex which is element of the same gadget, and to another vertex which is not element of the same gadget. For the crossover, $\{A, B, C, D\}$ are external edges; all other edges are internal. For the half-crossover, $\{a, b, c, d\}$ are external edges; all other edges are internal. It should be obvious that for



(a) Crossover (b) Half-crossover

Figure 2.2: Planar crossover gadgets, taken from [6].

the interaction with other gadgets, the orientation of the external edges is much more interesting than the orientation of the internal edges. We will also mention the definition of *geographically adjacent* edges. Consider a circle which crosses all external edges but none of the internal edges in our Euclidean embedding of the gadget. Any pair of edges that are connected by the circle, without crossing another edge in between, are considered geographically adjacent. For the half-crossover, $\{a, c\}$, $\{a, d\}$, $\{b, c\}$ and $\{b, d\}$ are geographically adjacent. Similar pairs of edges can be indicated at the crossover gadget.

The features of these gadgets can easily be verified, as done by the authors of [6]. They show that in the crossover gadget, A or B can be flipped outward if and only if the other of these two is pointing inward. They show a similar property for C and D. This implicitly means that at any time, there will be at least two of the external edges turned inward. Flipping an outward pointing external edge can eventually result in flipping the external edge which is not geographically adjacent to it.

As for the half-crossover, it can be verified that at any time there will be at least two of the external edges pointing inward. Flipping one of the outward pointing external edges can eventually result in flipping one of the inward pointing external edges. Regardless whether edges are already flipped, there are a total of six states with the property that two external edges are pointing outward. Of course, there are also states which do not satisfy this property. We will not analyse these as it can be verified that these will not occur in initial states of the crossover gadget. The six states are:

$$\begin{array}{lll} (a_{\text{out}}, b_{\text{out}}, c_{\text{in}}, d_{\text{in}}), & (a_{\text{out}}, b_{\text{in}}, c_{\text{out}}, d_{\text{in}}), & (a_{\text{out}}, b_{\text{in}}, c_{\text{in}}, d_{\text{out}}), \\ (a_{\text{in}}, b_{\text{out}}, c_{\text{out}}, d_{\text{in}}), & (a_{\text{in}}, b_{\text{out}}, c_{\text{in}}, d_{\text{out}}), & (a_{\text{in}}, b_{\text{in}}, c_{\text{out}}, d_{\text{out}}) \end{array}$$

A state of the crossover gadget is described as a tuple of the external edges and their orientation in subscript. Here, in means that the edge is pointing toward the gadget, out means that it is pointing away from the gadget. We supply an edge with a bar when it is already flipped, hence it can not be flipped again. From the orientations of these external edges, we can derive the orientation of most internal edges. For some internal edges, the orientation is undefined.

Chapter 3

Bounded NCL half-crossover

In this chapter we will analyse the half-crossover as used in the bounded version of NCL, and provide state space diagrams.

3.1 Internal state

Since the direction of some internal edges can not be derived from the external edges, we need a way to specify the direction of all edges. For this we will use the cardinal direction system, denoted in subscript. The state space of each edge is an element of this set:

$$\{\{\mathrm{N},\mathrm{S}\},\{\mathrm{W},\mathrm{E}\},\{\mathrm{NW},\mathrm{SE}\},\{\mathrm{NE},\mathrm{SW}\}\}$$

The state space of an edge is chosen depending on the embedding of the graph. If the edge is drawn vertical, it can point north (N) and south (S). If the edge is drawn horizontal, it can point west (W) and east (E). A similar statement can be made for the diagonal edges. We will call a description which specifies the orientation of all edges, including internal edges, the *internal state*.

Consider a half-crossover gadget which is in the following state:

$$(a_{\mathrm{in}}, b_{\mathrm{in}}, c_{\mathrm{out}}, d_{\mathrm{out}})$$

This implicitly means that edges e and f are pointing west; h is pointing south west; i is pointing north west; edges j and k are pointing east; n is pointing south east and finally o is pointing north east. Otherwise one of the

vertices would not satisfy its minimal inflow. The internal state is as follows:

$$(a_{\mathrm{S}}, b_{\mathrm{N}}, c_{\mathrm{W}}, d_{\mathrm{E}}, e_{\mathrm{W}}, f_{\mathrm{W}}, g, h_{\mathrm{SW}}, i_{\mathrm{NW}}, j_{\mathrm{E}}, k_{\mathrm{E}}, m, n_{\mathrm{SE}}, o_{\mathrm{NE}})$$

Note that the orientation of g and m is not specified in this internal state, as the inflows of these edges are of no influence to the constraints of the current state.

By flipping either c or d, we can flip either a or b. Formally stated, we could go to the following states: $(a_{\mathrm{out}}, b_{\mathrm{in}}, c_{\mathrm{in}}, d_{\mathrm{out}})$, $(a_{\mathrm{in}}, b_{\mathrm{out}}, c_{\mathrm{in}}, d_{\mathrm{out}})$, $(a_{\mathrm{out}}, b_{\mathrm{in}}, c_{\mathrm{out}}, d_{\mathrm{in}})$ and $(a_{\mathrm{in}}, b_{\mathrm{out}}, c_{\mathrm{out}}, d_{\mathrm{in}})$. As these are all symmetric, we will only show how to get to the first. After flipping c, h, g if needed and e, will result in the possibility to flip a. The current internal state of the gadget is now

$$(\overline{a_{\mathrm{N}}}, b_{\mathrm{N}}, \overline{c_{\mathrm{E}}}, d_{\mathrm{E}}, \overline{e_{\mathrm{E}}}, f_{\mathrm{W}}, g_{\mathrm{N}}, \overline{h_{\mathrm{NE}}}, i_{\mathrm{NW}}, j_{\mathrm{E}}, k_{\mathrm{E}}, m, n_{\mathrm{SE}}, o_{\mathrm{NE}})$$

or

$$(\overline{a_{\mathrm{N}}}, b_{\mathrm{N}}, \overline{c_{\mathrm{E}}}, d_{\mathrm{E}}, \overline{e_{\mathrm{E}}}, f_{\mathrm{W}}, \overline{g_{\mathrm{N}}}, \overline{h_{\mathrm{NE}}}, i_{\mathrm{NW}}, j_{\mathrm{E}}, k_{\mathrm{E}}, m, n_{\mathrm{SE}}, o_{\mathrm{NE}})$$

depending on whether g was already pointing north or not.

It is easy to see that from this state, we can find a sequence which will eventually also flip b. If we flip d, o, m if needed and k, it is possible to flip b. The internal state in which we now are is one of the following:

$$(\overline{a_{\mathrm{N}}}, \overline{b_{\mathrm{S}}}, \overline{c_{\mathrm{E}}}, \overline{d_{\mathrm{W}}}, \overline{e_{\mathrm{E}}}, f_{\mathrm{W}}, g_{\mathrm{N}}, \overline{h_{\mathrm{NE}}}, i_{\mathrm{NW}}, j_{\mathrm{E}}, \overline{k_{\mathrm{W}}}, m_{\mathrm{S}}, n_{\mathrm{SE}}, \overline{o_{\mathrm{SW}}})$$

$$(\overline{a_{\mathrm{N}}}, \overline{b_{\mathrm{S}}}, \overline{c_{\mathrm{E}}}, \overline{d_{\mathrm{W}}}, \overline{e_{\mathrm{E}}}, f_{\mathrm{W}}, \overline{g_{\mathrm{N}}}, \overline{h_{\mathrm{NE}}}, i_{\mathrm{NW}}, j_{\mathrm{E}}, \overline{k_{\mathrm{W}}}, m_{\mathrm{S}}, n_{\mathrm{SE}}, \overline{o_{\mathrm{SW}}})$$

$$(\overline{a_{\mathrm{N}}}, \overline{b_{\mathrm{S}}}, \overline{c_{\mathrm{E}}}, \overline{d_{\mathrm{W}}}, \overline{e_{\mathrm{E}}}, f_{\mathrm{W}}, g_{\mathrm{N}}, \overline{h_{\mathrm{NE}}}, i_{\mathrm{NW}}, j_{\mathrm{E}}, \overline{k_{\mathrm{W}}}, \overline{m_{\mathrm{S}}}, n_{\mathrm{SE}}, \overline{o_{\mathrm{SW}}})$$

$$(\overline{a_{\mathrm{N}}}, \overline{b_{\mathrm{S}}}, \overline{c_{\mathrm{E}}}, \overline{d_{\mathrm{W}}}, \overline{e_{\mathrm{E}}}, f_{\mathrm{W}}, \overline{g_{\mathrm{N}}}, \overline{h_{\mathrm{NE}}}, i_{\mathrm{NW}}, j_{\mathrm{E}}, \overline{k_{\mathrm{W}}}, \overline{m_{\mathrm{S}}}, n_{\mathrm{SE}}, \overline{o_{\mathrm{SW}}})$$

depending on the initial orientation of g and m. By the way, i and n can be flipped (with no effect whatsoever).

3.2 State spaces

An obvious question is which states can be reached from each state, and in which order. Again, we only consider states which have exactly two edges pointing inward and where none of the edges are flipped initially. States which have less inward edges can not be created without violating the NCL

rules; states which have more inward edges will not occur in initial states of the full crossover.

There are two different types of such a half-crossover, i.e., half-crossovers in which two geographically adjacent edges are pointing inward at the initial state, and we will refer to these as the *straight half-crossover*, and half-crossovers in which two geographically not adjacent edges are pointing inward at the initial state, and we will refer to this as the *bended half-crossover*. The straight half-crossovers can be in $(a_{\text{in}}, b_{\text{out}}, c_{\text{in}}, d_{\text{out}})$, $(a_{\text{in}}, b_{\text{out}}, c_{\text{out}}, d_{\text{in}})$, $(a_{\text{out}}, b_{\text{in}}, c_{\text{in}}, d_{\text{out}})$, $(a_{\text{out}}, b_{\text{in}}, c_{\text{out}}, d_{\text{in}})$ and all subsequent states; the bended half-crossovers can be in $(a_{\text{in}}, b_{\text{in}}, c_{\text{out}}, d_{\text{out}})$, $(a_{\text{out}}, b_{\text{out}}, c_{\text{in}}, d_{\text{in}})$ and all subsequent states.

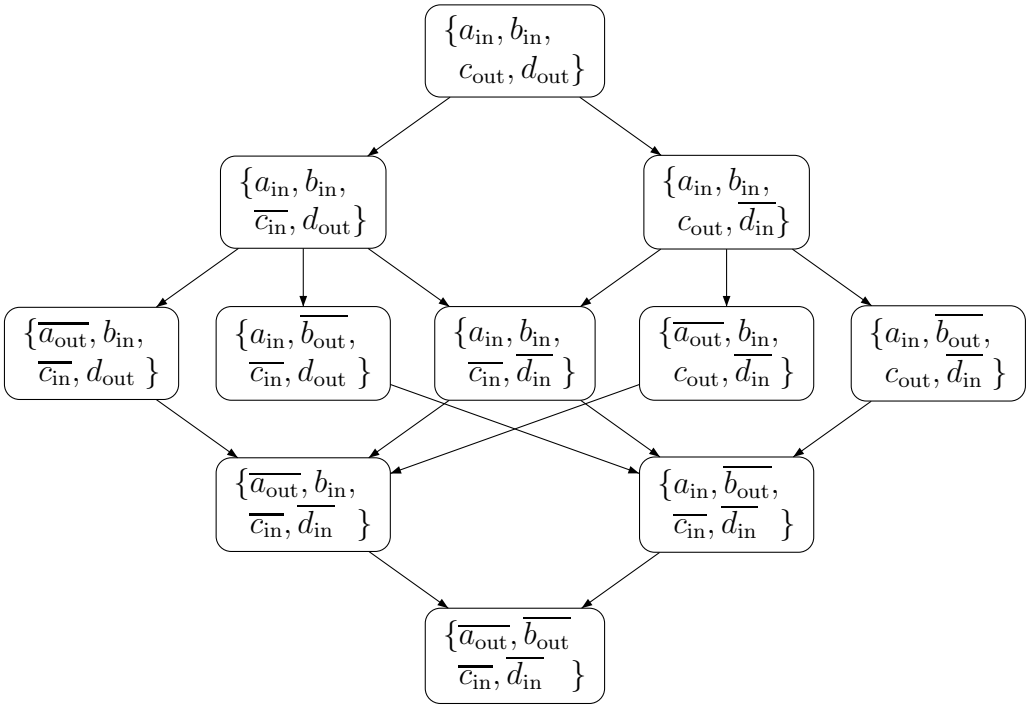


Figure 3.1: State space diagram of a bended half-crossover.

In Figure 3.1 the state space diagram of a bended half-crossover is shown. From this state space diagram we can observe two things:

1. In any state there will be at least two edges pointing inward. This is also proven in [6].

2. We can end up in a state where all four external edges are flipped.

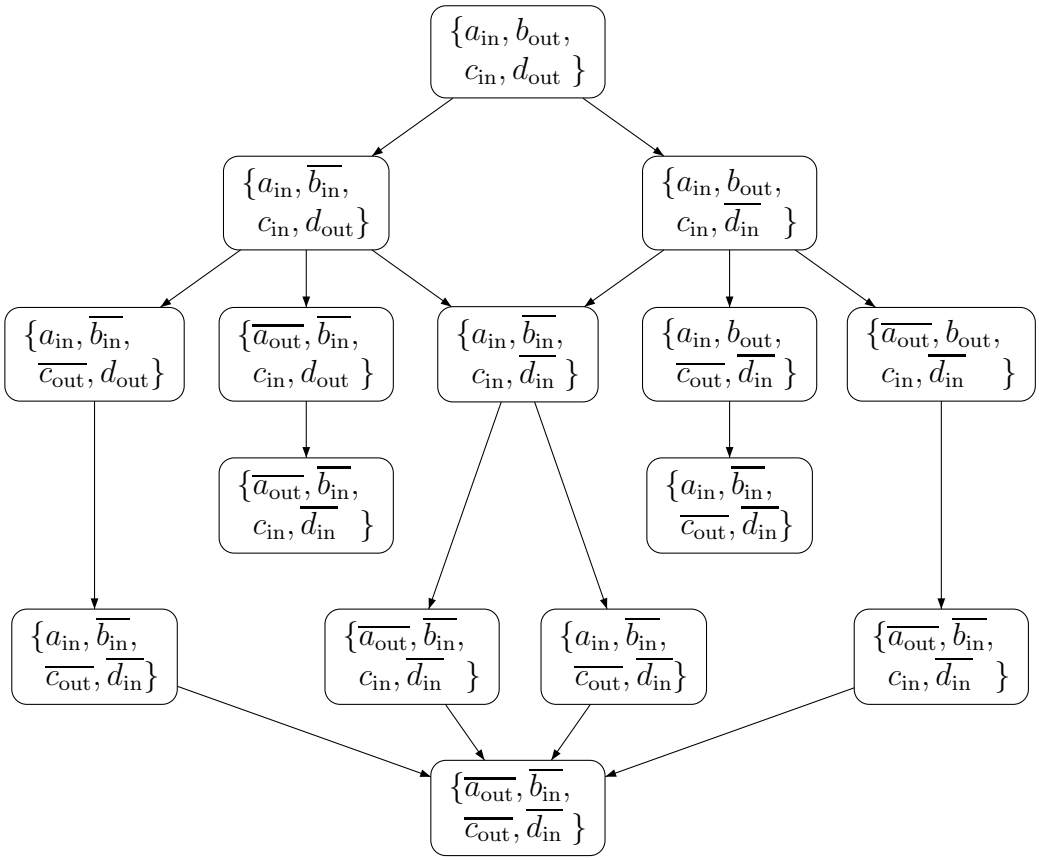


Figure 3.2: State space diagram of a straight half-crossover.

In Figure 3.2 a state space diagram of a straight half-crossover is shown. From this state space diagram we can also observe some things:

1. Again, at any time there will be at least two external edges pointing inward.

2. If we first flip two external edges which are geographically adjacent, we will always be able to flip the other two external edges.

3. If, starting from the initial configuration, we flip two edges which are geographically not adjacent, we will block the configuration and make it impossible for the last inward pointing edge to be flipped.

As for the last observation, this could have some consequences for the functionality of the crossover gadget. The correctness of this observation can be verified by checking the internal state. Initially, the internal state is one of the following

$$(a_{\mathrm{S}}, b_{\mathrm{S}}, c_{\mathrm{E}}, d_{\mathrm{E}}, e_{\mathrm{W}}, f_{\mathrm{E}}, g_{\mathrm{S}}, h_{\mathrm{NE}}, i_{\mathrm{SE}}, j_{\mathrm{E}}, k_{\mathrm{E}}, m_{\mathrm{S}}, n_{\mathrm{SE}}, o_{\mathrm{NE}})$$

$$(a_{\mathrm{S}}, b_{\mathrm{S}}, c_{\mathrm{E}}, d_{\mathrm{E}}, e_{\mathrm{W}}, f_{\mathrm{E}}, g_{\mathrm{S}}, h_{\mathrm{SW}}, i_{\mathrm{SE}}, j_{\mathrm{E}}, k_{\mathrm{E}}, m_{\mathrm{S}}, n_{\mathrm{SE}}, o_{\mathrm{NE}})$$

$$(a_{\mathrm{S}}, b_{\mathrm{S}}, c_{\mathrm{E}}, d_{\mathrm{E}}, e_{\mathrm{W}}, f_{\mathrm{E}}, g_{\mathrm{S}}, h_{\mathrm{NE}}, i_{\mathrm{SE}}, j_{\mathrm{E}}, k_{\mathrm{E}}, m_{\mathrm{N}}, n_{\mathrm{SE}}, o_{\mathrm{NE}})$$

$$(a_{\mathrm{S}}, b_{\mathrm{S}}, c_{\mathrm{E}}, d_{\mathrm{E}}, e_{\mathrm{W}}, f_{\mathrm{E}}, g_{\mathrm{S}}, h_{\mathrm{SW}}, i_{\mathrm{SE}}, j_{\mathrm{E}}, k_{\mathrm{E}}, m_{\mathrm{N}}, n_{\mathrm{SE}}, o_{\mathrm{NE}})$$

depending on the initial state of h and m. Assuming that h is already pointing north east and m is already pointing south, which is the most ideal scenario since one would have to flip less edges, the internal state after flipping b and a would be

$$(\overline{a_{\mathrm{N}}}, \overline{b_{\mathrm{N}}}, c_{\mathrm{E}}, d_{\mathrm{E}}, \overline{e_{\mathrm{E}}}, \overline{f_{\mathrm{W}}}, \overline{g_{\mathrm{N}}}, h_{\mathrm{NE}}, i_{\mathrm{SE}}, j_{\mathrm{E}}, k_{\mathrm{E}}, m_{\mathrm{S}}, n_{\mathrm{SE}}, o_{\mathrm{NE}})$$

After flipping d and c, rather than a and b, the internal state would be

$$(a_{\mathrm{S}}, b_{\mathrm{S}}, \overline{c_{\mathrm{W}}}, \overline{d_{\mathrm{W}}}, e_{\mathrm{W}}, \overline{f_{\mathrm{W}}}, g_{\mathrm{S}}, \overline{h_{\mathrm{SW}}}, \overline{i_{\mathrm{NW}}}, j_{\mathrm{E}}, \overline{k_{\mathrm{W}}}, m_{\mathrm{S}}, n_{\mathrm{SE}}, \overline{o_{\mathrm{SW}}})$$

It can be verified that from both internal states, it is impossible to also flip the last outward pointing edge. In the next chapter we will check which consequences this has for the crossover gadget.

Chapter 4

Bounded NCL crossover

In this chapter we will analyse the crossover gadget, which is shown in Figure 2.2(a).

4.1 State space

In [6] it is shown that A and B can not be simultaneously pointing outward, and neither can C and D. This means that the only legal states of the crossover gadget are

$$\begin{array}{ll} (A_{\text{out}}, B_{\text{in}}, C_{\text{out}}, D_{\text{in}}), & (A_{\text{out}}, B_{\text{in}}, C_{\text{in}}, D_{\text{out}}), \\ (A_{\text{in}}, B_{\text{out}}, C_{\text{out}}, D_{\text{in}}), & (A_{\text{out}}, B_{\text{in}}, C_{\text{in}}, D_{\text{out}}) \end{array}$$

Of course there are also states which have more than two inward pointing edges, but we will not analyse these. All states can be obtained from each other by rotation or reflection, hence all will have a similar state space. We will only show the state space diagram of $(A_{\text{in}}, B_{\text{out}}, C_{\text{in}}, D_{\text{out}})$. This can be seen in Figure 4.1. The initial internal state is

$$\begin{array}{ll} (& A_{\text{S}}, B_{\text{S}}, C_{\text{E}}, D_{\text{E}}, E_{\text{E}}, F_{\text{W}}, G_{\text{E}}, H_{\text{S}}, I_{\text{SE}}, J_{\text{NE}}, \\ & K_{\text{NE}}, L_{\text{SE}}, M_{\text{E}}, N_{\text{W}}, O_{\text{S}}, P_{\text{NE}}, Q_{\text{SE}}, R_{\text{SE}}, S_{\text{NE}} \quad) \end{array}$$

The presence of red-red-red-red vertices makes analysing this gadget somewhat different from analysing a gadget which does not contain red-red-red-red vertices. All used red-red-red-red vertices are straight half-crossovers, and as seen in Section 3.2 some sequences of moves are impossible. Besides checking for each edge that we want to flip whether the inflow constraint is not

violated and whether it has already been flipped or not, we must also check in which order the edges adjacent to a red-red-red-red vertex are flipped, because some of these orders might be impossible due to the properties of the straight half-crossover.

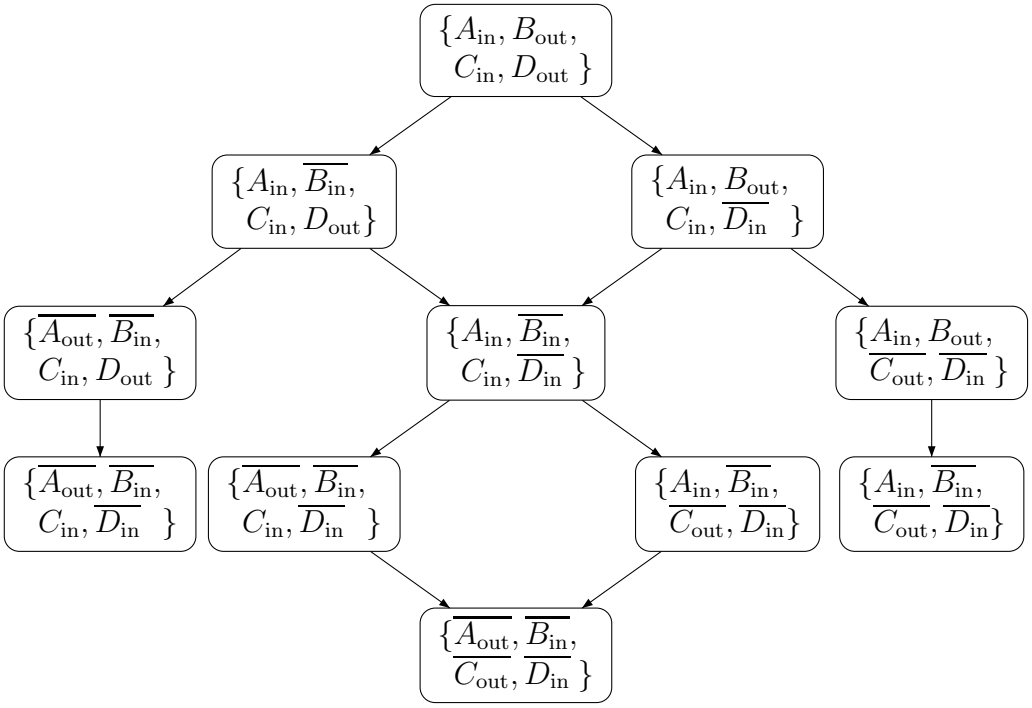


Figure 4.1: State-space diagram of an $(A_{\text{in}}, B_{\text{out}}, C_{\text{in}}, D_{\text{out}})$ crossover.

As suggested by the state space diagram, there is only one strategy to flip all edges: First flip both outward pointing edges in some order, afterwards flip both inward pointing edges in some order. When flipping an inward pointing edge before both outward pointing edges are flipped, will result in the impossibility to flip the other inward pointing edge. We will say that this gadget does not have the *Full Crossing Property*, abbreviated as the FCP.

4.2 Full Crossing Property

In this section we will give a definition of the Full Crossing Property and show why it is impossible to construct a gadget in the restricted version of NCL which has the full crossing property. This concept relies on the fact that each legal NCL Graph has a maximal degree of three, and that each edge may only be reversed once in the restricted version.

A gadget which has the full crossing property, has two sets (both are of size 2) of geographically non adjacent edges and obeys the following rules:

1. at all time, there must be at least one element of the set pointing inward;

2. if there is an element pointing outward, reversing this must allow the other element to point outward after a sequence of moves;

3. if there is no element pointing outward, for each element there must be a sequence of moves that eventually turns it outward. Of course, in the restricted version doing one of these move sequences excludes all others.

Note that the gadget from Figure 2.2(a) does not have the FCP, as it does not obey the second rule. After a sequence which flips the edges of one of the two sets, the outward pointing edges in the other set can not be flipped anymore.

As for the proof that it is impossible to create a gadget with the FCP, consider two sets of geographically non adjacent edges ($\{A, B\}$ and $\{C, D\}$) that need to cross each other. When reversing an edge that is pointing outward (A), in order to reverse the corresponding edge (B, which is pointing inward at that time) a group of adjacent edges between A and B must be reversed in some order. We will refer to these edges as the *split edges*. We want to stress the fact that an uninterrupted set of edges between A and B will all be reversed; if this was not the case, then B could also be reversed without reversing A, which would break the first rule.

For the other set of connected edges a similar group of split edges must be defined. However it is forbidden to reverse any of the edges that we already used for reversing A and B. It is impossible to define such a set from one side of the previous split edges to the other side. The only legal position to cross is at a vertex. Since all vertices have a maximal degree of three, from which

two are already occupied by the split edges of A and B, there is no way to define a new set of split edges which do not use any of the edges that are already reversed. We conclude that it is impossible to create a gadget with the full crossing property for Bounded NCL.

4.3 Acyclic Bounded NCL

In this section we will argue why NCL does not need a gadget with the FCP. We define *Acyclic Bounded NCL* to be the subset of Bounded NCL, containing all acyclic Bounded NCL graphs consisting of AND, OR, FANOUT and CHOICE vertices (shown in Figure 2.1). Every instance of 3-SAT can be reduced to an instance of Acyclic Bounded NCL using a similar reduction as the one to Bounded NCL, as shown in [6]. A diagram of this hierarchy is shown in Figure 4.2.

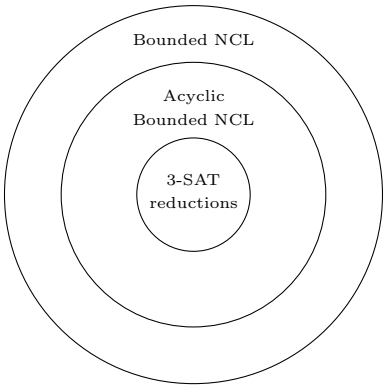


Figure 4.2: Hierarchy of Bounded NCL subsets.

Every instance of 3-SAT is a (Boolean) formula. This formula is translated into a NCL graph in a straightforward way, ensuring the graph is always acyclic, see Figure 4.3. Central in the graph are vertices representing the clauses. These are joined to the topmost target edge using AND vertices. At the bottom of the graph for the formula we have one vertex for each variable. CHOICE vertices ensure that for each variable we cannot set both that variable and its negation. The values of the variables are duplicated for the various occurrences in the clauses using FANOUT. Then (copies of) the variables are attached to the proper clauses using OR vertices, thus representing the semantics of the formula. Obviously the resulting graph is acyclic.

We can reduce Acyclic Bounded NCL to Planar Bounded NCL, by replacing every crossing in an instance by the construction of Figure 2.2(a). As this graph is acyclic in terms of edge direction, the problem mentioned in Section 4 does not apply here.

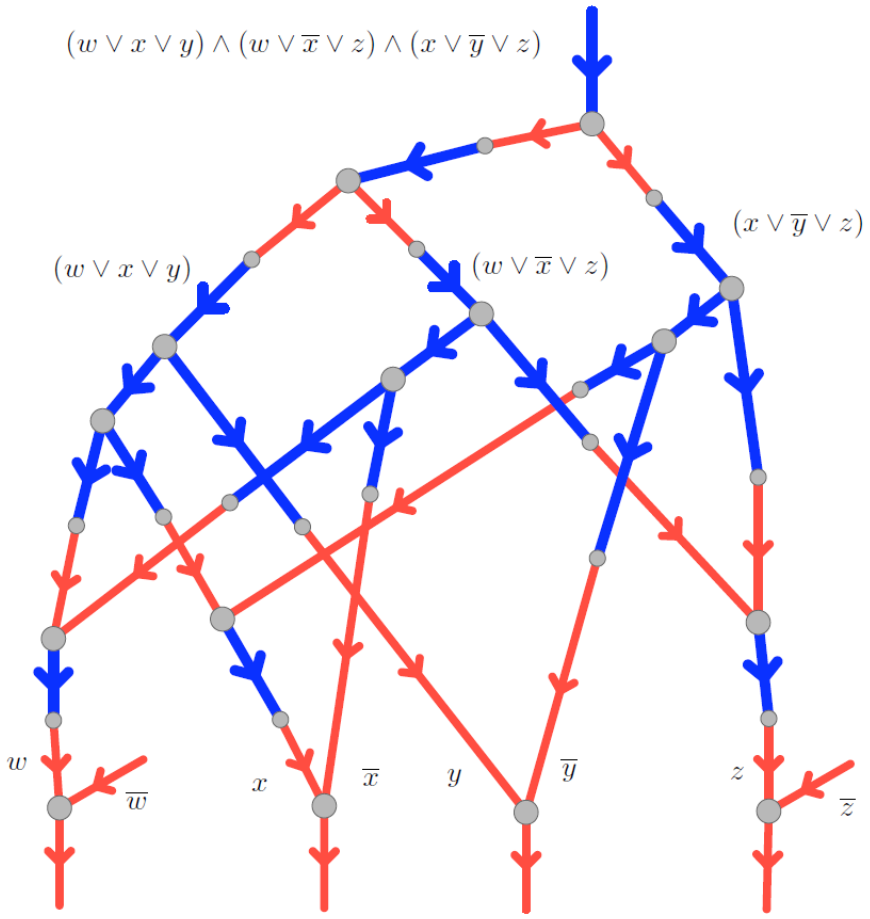


Figure 4.3: A constraint graph corresponding to the formula $(w \vee x \vee y) \wedge (w \vee \overline{x} \vee z) \wedge (x \vee \overline{y} \vee z)$, taken from [6].

Chapter 5

Klondike

Klondike is a well-known solitaire card game, popularized by Microsoft Windows. The normal version of the game is played with a standard French card deck, without jokers. A typical instance of Klondike is shown in Figure 5.1. The authors of [9] have given a formal definition of the game and provided an algorithm that plays Klondike games with a high success rate. The authors of [7] show, amongst other complexity results, that Klondike is NP-complete even when played with three suits. We will give a formal definition of Klondike and confirm the NP-completeness of Klondike by a reduction from Bounded NCL.

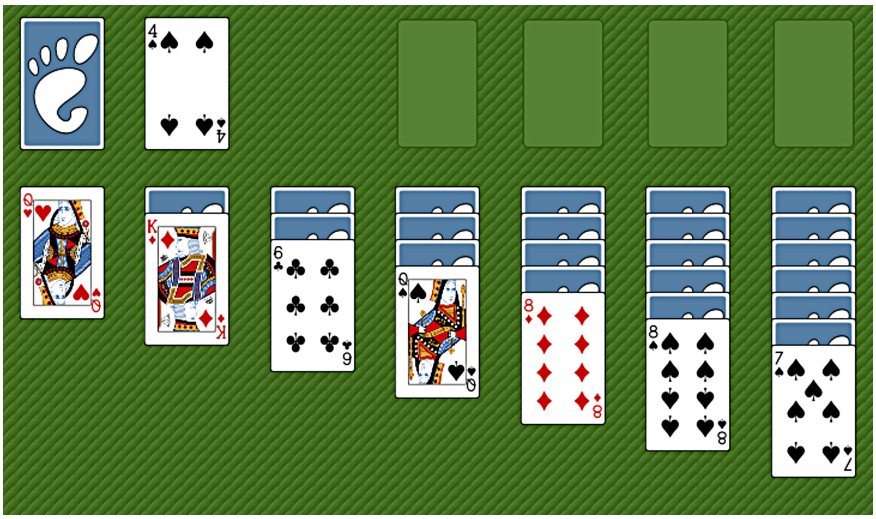


Figure 5.1: A typical instance of Klondike.

5.1 Definition

Klondike is played with a card deck containing m suits, numbered from 1 to m, each suit containing n cards ranked from 1 to n. A card with rank 1 is also referred to as an *Ace*; a card with rank $n-2$ is also referred to as a *Jack*; a card with rank $n-1$ is also referred to as a *Queen* and finally a card with rank n is also referred to as a *King*. The functions $rank(c)$ and $suit(c)$ return the rank of card c and the suit of card c, respectively. Each suit is colored either red or black. The function $color(s)$ returns the color of suit s.

A Klondike game consists of m *suit stacks*, k *build stacks*, a *pile stack* and a *talon*. A stack S is defined to be a list of cards, where $0 \leq |S| \leq n$. A *configuration* describes for each card in which stack it is and on which position. For every card in a build stack it also describes whether the card is *face-up* or *face-down*. The subset of cards that are turned faced-up on a certain build stack forms together a *card block*. In an initial configuration each build stack b, where $b \in \{1, 2, 3, \ldots, k-2, k-1, k\}$, contains b cards, all face-down; all other cards are in the pile stack.

We will define the notion of *acceptance*, which determines which moves the player can make. Each suit stack that is empty accepts card c if and only if c has rank 1 (Ace). Every suit stack that is not empty, containing card t on top, accepts card c if and only if $suit(c)$ equals $suit(t)$, and $rank(c)$ equals $rank(t) + 1$. Each card block that is not empty, containing card t on top, accepts card c if and only if $color(suit(t))$ does not equal $color(suit(c))$ and $rank(c)$ equals $rank(t) - 1$.

On each turn, the player can move cards from one stack to another in the following manner:

1. If all cards on a build stack are face-down, the card op top can be turned face-up. Note that a card block containing this card is created.

2. The top card of a card block can be moved to the top of a suit stack, provided that the suit stack accepts this card.

3. A whole card block p can be moved to the top of another card block q, provided that q accepts the card at the bottom of p. In some versions of the game a partial card block can also be moved to another card block, subjected to a similar acceptance rule. For our purposes it does not matter which of these versions is applied.

4. If the pile stack is not empty, the top p cards can be moved to the talon, which maintains its cards in a first-in-last-out order. Here, p is defined to be $min(3, |p|)$.

5. The top card of the talon can be moved to either a suit stack or a build block, if the corresponding stack accepts this card.

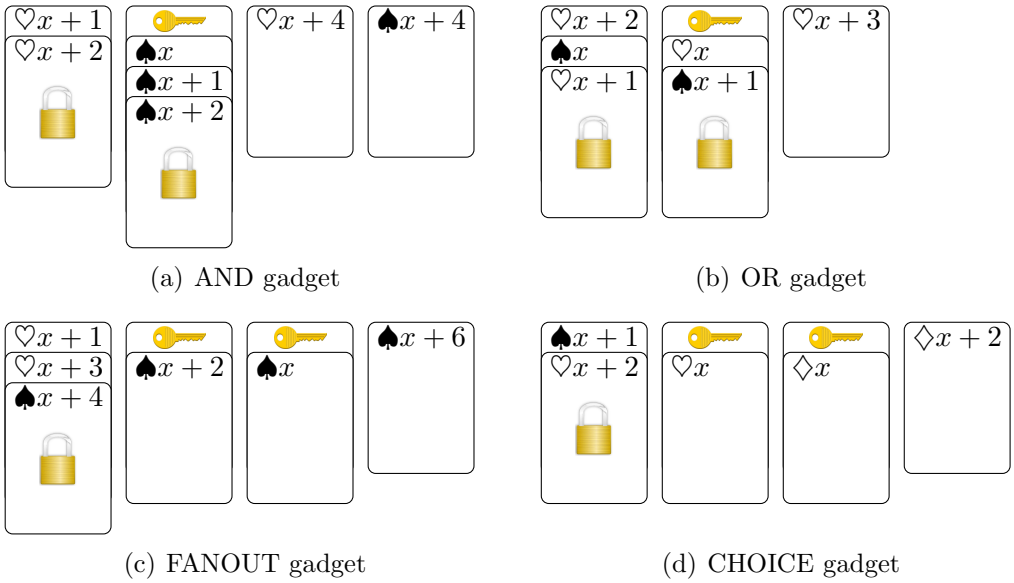
6. If the pile stack is empty, all cards on the talon can be moved to the pile stack. The order of the cards must be preserved.

The goal is to move all cards to the suit stacks, when this is achieved the player has won.

5.2 NP-completeness

We will show that Klondike is NP-complete, even when played with two black and two red suits. We reduce from Bounded NCL. We will use the four suits from the French card deck, which are black spades (♠), black clubs (♣), red hearts (♡) and red diamonds (♢). We will show that every Bounded NCL graph can be transformed into a Klondike configuration, such that the Klondike game can be won if and only if the target edge of the Bounded NCL graph can be flipped.

Every vertex in the original Bounded NCL graph can be replaced by one of the gadgets in Figure 5.2. All gadgets consist of three or four build stacks, with all cards face-down. Each gadget gets a range of unique ranks assigned to it, x to $x+10$. (The cards with ranks within this range that are not used in this gadget will be placed in a so-called *victory gadget*, which we will explain further on.) All cards in this gadget, except the *key card*, will have a rank within this range. A *lock cards* represents the tail of an edge adjacent to the corresponding Bounded NCL vertex; a lock icon is displayed on these cards. A key cards represents the head of an edge adjacent to the corresponding Bounded NCL vertex; a key icon is displayed on these cards. The suit and rank of the key card are assigned such that once turned face-up, it accepts a lock card of the gadget from which the corresponding Bounded NCL edge is pointing. The gadgets are constructed in such a way that the key card can be turned face-up if and only if the corresponding Bounded NCL edge can be flipped. The usage of the card blocks containing just one card will be explained further on.



(a) AND gadget (b) OR gadget

(c) FANOUT gadget (d) CHOICE gadget

Figure 5.2: Klondike gadgets.

For each lock card ℓ it is easy to see which card should be turned face-up in order to move it. If $color(rank(\ell))$ is red, these cards are $\spadesuit rank(\ell)+1$ and $\clubsuit rank(\ell)+1$. Otherwise these cards are $\heartsuit rank(\ell)+1$ and $\diamondsuit rank(\ell)+1$. Only one of these cards is used in another gadget, where it serves as a key card. The other card is defined to be a *complementary key card*, and the use of these cards is explained further on.

In the original Klondike configuration the pile stack, talon and suit stacks are completely empty. The player can obtain the Aces, which are necessary in order to move cards to the suit stacks, if and only if the card which represents the target edge can be turned face-up. This will initiate the end game, which is always winning for the player as we will show further on.

Lemma 5.2.1 *The construction in Figure 5.2(a) satisfies the same constraints as a Bounded NCL AND vertex, with* $\heartsuit x+2$ *and* $\spadesuit x+2$ *corresponding to the input edges, and the key card to the output edge.*

Proof In order to turn the key card face-up, both $\heartsuit x+2$ and $\spadesuit x+2$ must be moved to another card block. If neither of these cards can be moved, no moves can be done within this gadget. If we can only move $\heartsuit x+2$, we can move $\heartsuit x+1$ to the card block consisting of $\spadesuit x+2$, but after that no other

moves can be done. If we can only move $\spadesuit x + 2$ a similar argument can be made. If we can move both $\heartsuit x + 2$ and $\spadesuit x + 2$ to another card block, we can also move $\heartsuit x + 1$ and $\spadesuit x + 1$ to the card blocks with $\spadesuit x + 2$ and $\heartsuit + 2$ on top respectively, and finally $\spadesuit x$ to the card block where now $\heartsuit x + 1$ is on top. At this point we can turn the key card face-up.

Lemma 5.2.2 *The construction in Figure 5.2(b) satisfies the same constraints as a Bounded NCL OR vertex, with* $\heartsuit x+1$ *and* $\spadesuit x+1$ *corresponding to the input edges, and the key card to the output edge.*

Proof In order to turn the key card face-up, $\heartsuit x+1$ or $\spadesuit x+1$ must be moved to another card block. If neither of these cards can be moved, no moves can be done within this gadget. If we can move $\spadesuit x + 1$ to another card block, $\heartsuit x$ can be moved to this card block. We can now turn the key card face up. If we can move $\heartsuit x + 1$ to another card block, $\spadesuit x$ can be moved to this card block. We can now turn $\heartsuit x + 2$ face-up, and move $\spadesuit x + 1$ to this card block. After this $\heartsuit x$ can be moved to this card block. At this point we can turn the key card face-up. Note that some cards of this gadget may remain face-down depending on which lock card is used. We will take care of these cards later.

Lemma 5.2.3 *The construction in Figure 5.2(c) satisfies the same constraints as a Bounded NCL FANOUT vertex, with* $\spadesuit x + 4$ *corresponding to the input edge, and the key cards to the output edges.*

Proof In order to turn both key cards face-up, $\spadesuit x + 4$ must be moved to another card block. If this card can not be moved, no moves can be done within this gadget. If we can move $\spadesuit x + 4$ to another card block, we can move in order $\heartsuit x + 3$, $\spadesuit x + 2$, $\heartsuit x + 1$ and $\spadesuit x$ to this card block. We can now turn both key cards face-up.

Lemma 5.2.4 *The construction in Figure 5.2(d) satisfies the same constraints as a Bounded NCL CHOICE vertex, with* $\heartsuit x + 2$ *corresponding to the input edge, and the key cards to the output edges.*

Proof In order to turn one of the key cards face-up, $\heartsuit x + 2$ must be moved to another card block. If this card can not be moved, no moves can be done within this gadget. If we can move $\heartsuit x + 2$ to another card block, $\spadesuit x + 1$ can be moved to this card block afterward. Now we can move either $\heartsuit x$ or $\diamondsuit x$ to this card block. One of the key cards can now be turned face-up. Note that the only gadget that consists of a $\diamondsuit$ card is this one.

Theorem 5.2.5 *Klondike is NP-complete.*

Proof Reduction from Bounded NCL. Given a constraint graph made of AND, OR, FANOUT and CHOICE vertices, we construct a corresponding Klondike configuration using the gadgets shown in Figure 5.2. We also need a way to ensure that the player can move all cards to the suit stacks if and only if the key card corresponding to the target edge can be turned face-up. For this we have created a *victory gadget*. The victory gadget consists of a FANOUT, from which the lock card can be moved if and only if we can turn the key card corresponding to the target edge face-up. The keys of the FANOUT gadget open two other stacks: a stack with all complementary key cards in it, and a stack which contains all other cards which are not present in any other stack, ordered ascending by rank. For each rank the cards are given in ♣♢♡♠ order. (In fact, this order is of no importance.) The player can now move all complementary key cards to their associated gadgets (note that it is guaranteed that the complementary key cards are accepted by one of the build stacks, this is due to the stacks consisting of one card) and move the lock cards of those gadgets from which some or all of the lock cards were not yet moved. This ensures that all cards on the build stacks can be turned face-up and placed on build blocks in descending order. Now this is done the player can start building the suit stacks, starting with the Aces and eventually finishing with the Kings.

For creating the Klondike configuration, the number of cards and stacks we need are both bounded by a linear function over the number of vertices in the corresponding Bounded NCL graph. Any card which is used will be moved at most twice. Therefore Klondike is in NP, since any potential solution can be verified in polynomial time.

Chapter 6

Mahjong Solitaire

Mahjong Solitaire, also known as Shanghai Solitaire, is a one player puzzle game mainly played on the computer in which the player a randomly arranged stack of tiles is presented. A typical instance of Mahjong is shown in Figure 6.1. The goal is to remove all tiles in matching pairs of two. The authors of [3] have given a formal definition of this game, and have shown that a version of this game with imperfect information is PSPACE-complete. The author of [5] has stated a proof that a version of this game with perfect information is NP-complete. We will give a formal definition of this game and validate the latter result by a reduction from Bounded NCL.

Figure 6.1: A typical instance of Mahjong Solitaire.

6.1 Definition

For describing the game Mahjong Solitaire, we allow ourselves to use a modified version of the definition provided by the authors of [3].

The game uses Mahjong tiles (comparable with cards), that are divided into m disjoint *tile sets* $\mathcal{T}_p$ of $|\mathcal{T}_p| = s_p$ matching tiles, where s_p is an even number ($p = 1, 2, \ldots, m$). We define the set of all tiles to be $\mathcal{T} = \bigcup_p \mathcal{T}_p$. Tiles a and b *match*, if and only if for some p it holds that $a, b \in \mathcal{T}_p$. We generalize the standard game simply by assuming that there is an arbitrarily large, finite number of tiles. A *configuration* C is a set of positions (i, j, k), where each of i, j, k is a non-negative integer, satisfying the following constraints:

1. If $(i, j, k) \in C$ and $(i, j', k) \in C$ where $j < j'$, then for every j'' in the range $[j, j']$, $(i, j'', k) \in C$.

2. If $(i, j, k) \in C$ where $k > 0$ then $(i, j, k - 1) \in C$.

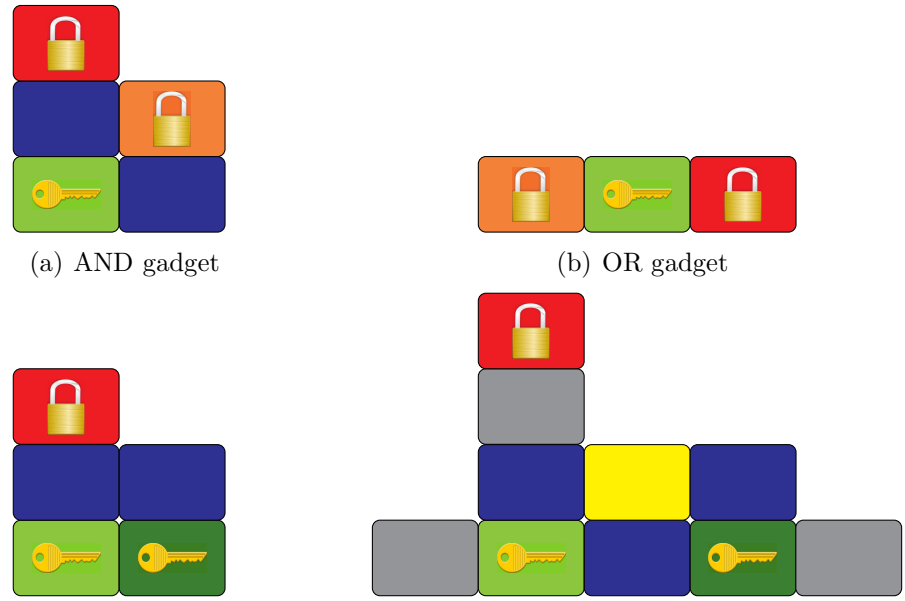
Intuitively, this captures the fact that tiles are arranged in three dimensions. Tiles can be stacked on top of each other; all tiles with common k are at the same height. All tiles with a common i index, form a *cross section*. Tiles at the same height, with common i index, form a *row*. The first condition ensures that there cannot be gaps in a row; the second, that a tile at height $k > 0$ must have a tile underneath it (in fact, at position $(i, j, k - 1)$).

With respect to a given configuration, a position (i, j, k) is *hidden* if in the configuration also a position $(i, j, k + 1)$ exists; the other positions are called *visible*. An *arrangement* consists of a set of tiles $\mathcal{T}$, a configuration C of size $|\mathcal{T}|$, and a 1-1 function f from the positions of C to $\mathcal{T}$. Intuitively, this means that $f(i, j, k)$ is the tile at position (i, j, k). If this function maps position (i, j, k) to tile t we say $pos(t) = (i, j, k)$. The elements of $\mathcal{T}$ will be mapped to the elements of C in such a way, that every possible combination is equally likely. With respect to a given arrangement, we say a position (i, j, k) is *available* if it is not hidden, and either position $(i, j - 1, k) \notin C$ or position $(i, j + 1, k) \notin C$ or both. An arrangement is called empty if $\mathcal{T}$ is empty.

A legal move consists of the removal of two matching tiles a, b that are both available. Formally, $\mathcal{T}' = \mathcal{T} - \{a, b\}$ and $C' = C - \{a_{pos}, b_{pos}\}$. The game is won if a series of moves results in the empty arrangement.

6.2 NP-completeness

We will shown that Mahjong Solitaire is NP-complete, even if for all p it holds that $s_p \leq 4$. We reduce from Bounded NCL. Each vertex in the original Bounded NCL graph can be replaced by one of the gadgets in Figure 6.2. These gadgets consist of one cross section, in which one or multiple rows are stacked. In each gadget, tiles that have the same color are element of $\mathcal{T}_p$, for some p.



(a) AND gadget (b) OR gadget

(c) FANOUT gadget (d) CHOICE gadget

Figure 6.2: Mahjong gadgets.

Every gadget used in the Mahjong arrangement has a set of numbers Q assigned to it, from the interval $[1, m]$. Every number is assigned to exactly one gadget. Each tile used in a gadget is element of $\mathcal{T}_q$, where $q \in Q$, except the tiles on which a key icon is displayed. A *lock tile* represents the tail of an edge adjacent to the corresponding Bounded NCL vertex; a lock icon is displayed on these tiles. Note that all lock tiles are already available. A *key tile* represents the head of an edge which is adjacent to the corresponding Bounded NCL vertex; a key icon is displayed on these tiles. A key tile is element of $\mathcal{T}_q$, with q being the same as a lock tile from the gadget from which the corresponding Bounded NCL edge is pointing.

When a key tile is available, it can be removed together with a lock tile from one of the other gadgets. The target edge in the corresponding Bounded NCL graph is always represented by a key tile. When this target edge is available this will initiate the end game, which is always winning for the player as we will show further on.

Lemma 6.2.1 *The construction in Figure 6.2(a) satisfies the same constraints as a Bounded NCL AND vertex, with the orange and red lock tiles corresponding to the input edges, and the green key tile corresponding to the output edge.*

Proof In order to get the key tile available, both lock tiles need to be removed. If neither of these can be removed, no moves can be done within this gadget. If only the red lock tile can be removed, no other moves can be done within this gadget. A similar argument can be given for the orange lock tile. If we can remove both lock tiles, both blue tiles are available and can be removed together. After this the green key tile is available.

Lemma 6.2.2 *The construction in Figure 6.2(b) satisfies the same constraints as a Bounded NCL OR vertex, with the orange and red lock tiles corresponding to the input edges, and the green key tile corresponding to the output edge.*

Proof In order to get the key tile available, at least one of the lock tiles needs to be removed. If neither of these can be removed, no moves can be done within this gadget. If the red lock tile can be removed, the green key tile is available. A similar argument can be given for the orange lock tile.

Lemma 6.2.3 *The construction in Figure 6.2(c) satisfies the same constraints as a Bounded NCL FANOUT vertex, with the red tile corresponding to the input edge, and the green and dark green key tiles corresponding to the output edges.*

Proof In order to get the key tiles available, the lock tile needs to be removed. As long as the lock tile is not removed, no other moves can be done within this gadget. When the lock tile is removed, both blue tiles can be removed together after which both key tiles are available.

Lemma 6.2.4 *The construction in Figure 6.2(d) satisfies the same constraints as a Bounded NCL CHOICE vertex, with the red tile corresponding to the input edge, and the green and dark green key tiles corresponding to the output edges.*

Proof In order to get a key tile available, the lock tile needs to be removed. Without the lock tile removed, we can only remove two of the gray tiles, after this no other moves are possible. If the lock tile is removed, we can match the newly available gray tile with either one of the other gray tiles. Both blue tiles can be removed, and depending on which gray tile was removed the green or the dark green key tile is now available. It is impossible to get both key tiles available.

Theorem 6.2.5 *Mahjong Solitaire is NP-complete.*

Proof Reduction from Bounded NCL. Given a constraint graph made of AND, OR, FANOUT and CHOICE vertices, we construct a corresponding Mahjong Solitaire arrangement using the gadgets shown in Figure 6.2. We can construct so-called *victory gadgets*, which will ensure that upon removing the key tile corresponding with the target edge, all other tiles can be removed. For this the set of *victory tiles* V is defined, which contains for every key tile and lock tile used in the other gadgets, a tile which is element of the same tile set. Besides these tiles, it contains for every CHOICE gadget used i) a tile which is element of the same tile set as the gray colored tile ii) a tile which is element of the same tile set as the blue colored tile and iii) a tile that is element of the same tile set as the yellow colored tile. We need to construct v victory gadgets, v being $\lceil |V|/2 \rceil$. Each victory gadget i, with $i \in \{1, 2, \ldots, v-1\}$, consists of a row containing 6 tiles, as shown in Figure 6.3. The numbers displayed on a tile indicate to which tile set it belongs. (Victory gadget v is somewhat similar to this, it lacks the tiles ℓ_{i+1} and r_{i+1}. If $|V|$ is odd, it also lacks tile v_i^2.) Here, v_i^1 and v_i^2 are both tiles from V, chosen in such a way that each tile from V will occur in exactly one victory gadget. The tiles ℓ_{i+1} and r_{i+1} are key tiles, these can be removed together with the leftmost and rightmost tile from the next victory gadget. From this it follows that ℓ_i and r_i are lock tiles, these will be removed together with the key tiles from the previous gadget. The key tile corresponding with the target edge will be connected to a FANOUT gadget, in which one of the key tiles is from the same tile set as ℓ_1 and the other key tile is of the same tile set as r_1. This

Figure 6.3: Mahjong victory gadget

ensures that if and only if we can get the key tile corresponding to the target edge available, all victory gadgets can be opened and all tiles from set V will be available. It can be verified that when having all tiles from V available, the player can always remove all remaining tiles, including the remaining tiles of the CHOICE gadgets. Gadgets from which some of the lock tiles were not yet removed can now have their remaining lock tiles removed.

For creating this Mahjong Solitaire, both the number of tiles and the number of tile sets we need to use is linearly bounded by the number of vertices used in the corresponding Bounded NCL graph. Therefore Mahjong Solitaire is in NP, since any potential solution can be verified in polynomial time.

Chapter 7

Nonograms

A *Nonogram*, also referred to as a Japanese puzzle, is a logic puzzle which can be considered as an image reconstruction problem. The player is presented a rectangular grid; for each row and column a description consisting of one or more integers is provided, representing consecutive cells that need to be colored. If the player can color a subset of cells in such a way that it is consistent with the description of all rows and columns, he has solved the puzzle and won the game. An example of a Nonogram and its solution is shown in Figure 7.1.

		1	5	2	5	2 1	2
2	1						
1	3						
1	2						
	3						
	4						
	1						

(a) 6×6 Nonogram

		1	5	2	5	2 1	2
2	1	■	■				■
1	3		■		■	■	■
1	2		■		■	■	
	3		■	■	■		
	4		■	■	■	■	
	1				■		

(b) Solved Nonogram

Figure 7.1: A Nonogram (a) and its unique solution (b), taken from [1].

The authors of [1] have given a formal definition of Nonograms and provided an algorithm for solving many Nonograms in polynomial time. In [8] it is proven that the Another Solution Problem for Nonograms in NP-complete.

We will give a formal definition of Nonograms and show that the decision question whether there exists a solution is NP-complete, by reduction from Planar Bounded NCL.

7.1 Definition

A Nonogram is a puzzle in which the player is presented an $m \times n$ grid, consisting of m rows and n columns. The rows are numbered from 1 through m, inclusive; the columns are numbered from 1 to n, inclusive. Each intersection between a row and a column is called a *cell*. The state of a cell is either *blank* or *colored*. Initially, all cells are *blank*. The function $\mathcal{C}(r, c)$ maps to the cell in row r and column c, with $1 \leq r \leq m$ and $1 \leq c \leq n$. A *line* is defined to be either a row or a column; the grid consists of ℓ lines, where $\ell = m + n$.

For each line i (with $1 \leq i \leq \ell$) a description d^i is provided, d^i being an ordered series of integers $(d^i_1, d^i_2, \ldots, d^i_{k_i})$. The function $R(r)$ returns d^r, being the description of the r^{th} row. The function $C(c)$ returns $d^{(m+c)}$, being the description of the c^{th} column.

The player is challenged to color a subset of cells, in such a way that for each line i, there are exactly k_i *colored segments*, where each colored segment s is of size d^i_s $(s = 1, 2, \ldots, k_i)$. A colored segment is defined to be a group of consecutive cells on an interval of a line, such that it holds that all cells within the interval are *colored*, and both cells adjacent to the interval, if any, are *blank*. If the player succeeds in doing so, he has won the game.

It can be easily verified that there exist Nonograms for which there is no subset of cells that be *colored* such that the constraints are fulfilled. In this case the player can not win. There are also Nonograms for which there are multiple subsets of cells that can be *colored* such that the constraints are complied. For more information about these cases, the reader is referred to [1].

7.2 NP-completeness

We will show that solving Nonograms is NP-complete, by reduction from Planar Bounded NCL. The global layout of the construction will be as shown in Figure 7.2. There will be several groups of D adjacent columns (or rows) whose description consists of a single element, i.e., m (or n), such that the

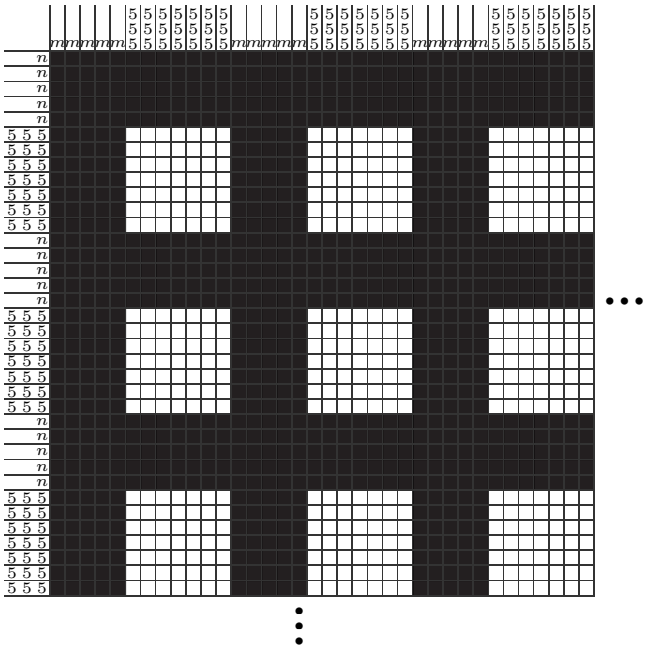


Figure 7.2: Global layout.

pattern of Figure 7.2 is maintained. We will call these lines the *separation lines*. Between each group of separation lines, there are G other lines. In the case of Figure 7.2, $D = 5$ and $G = 7$. The period P is defined to be $D + G$, in this case 12.

Formally, each column c with $1 \leq c \leq n$ is a separation line if and only if $(c - 1) \bmod P < D$. Also, each row r with $1 \leq r \leq m$ is a separation line if and only if $(r - 1) \bmod P < D$. The description of all other rows and columns should be consistent with this pattern. We will consider each element of any description bigger than or equal to D a *delimiter*. This means that the description of each other column should at least contain m/P delimiters; the description of each other row should at least contain n/P delimiters.

As a result of this pattern we can specify disjoint *subnonograms* between the separation lines. A subnonogram is defined to be a part of the nonogram, surrounded by separation lines. The function $S(s_y, s_x)$ refers to the subnonogram embedded in the rectangle between the cells $\mathcal{C}(s_y \times 12 + 6, s_x \times 12 + 6)$ and $\mathcal{C}(s_y \times 12 + 12, s_x \times 12 + 12)$, inclusive. We can insert the description associated with subnonogram $S(s_y, s_x)$ between the $(s_x)^{th}$ and $(s_x + 1)^{th}$ de-

limiter on the associated columns, and $(s_y)^{th}$ and $(s_y + 1)^{th}$ delimiter on the associated rows.

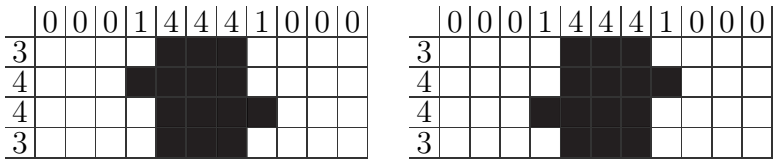


Figure 7.3: The two solutions of a Nonogram featuring two subnonograms horizontally separated by 3 separation lines.

It is also possible to send a signal between two orthogonal adjacent subnonograms by slightly adjusting delimiters between them. This is illustrated in Figure 7.3. The figure shows two subnonograms separated by $D = 3$ separation lines; one subnonogram between cells $\mathcal{C}(1, 1)$ and $\mathcal{C}(4, 4)$, inclusive, and one subnonogram between cells $\mathcal{C}(1, 8)$ and $\mathcal{C}(4, 11)$, inclusive. If we were to decide that $\mathcal{C}(3, 4)$ should be *colored*, this would explicitly mean that cell $\mathcal{C}(3, 8)$ can not be *colored*. (Note that this would also explicitly mean that $\mathcal{C}(2, 8)$ is *colored*, and $\mathcal{C}(2, 4)$ is not *colored*.) The opposite is also true. We will use this property to construct gadgets within a subnonogram, and propagate signals between them. This way we can embed a Planar Bounded NCL graph on a Nonogram grid.

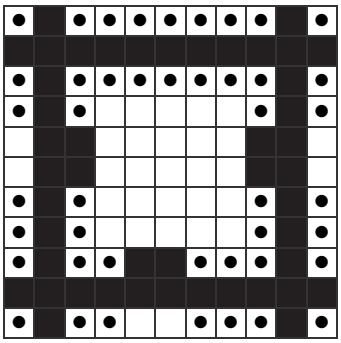
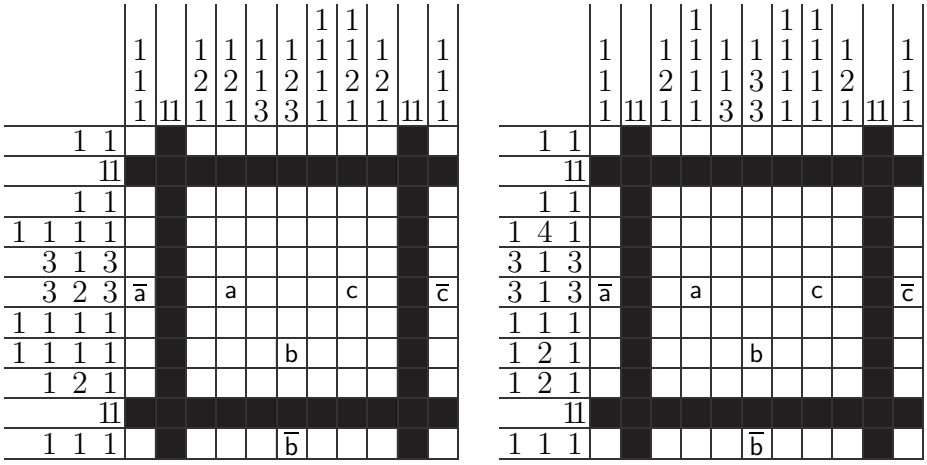


Figure 7.4: Template for Nonogram gadgets.

In Figure 7.2 a template of the gadgets is shown. From the description of every gadget follows immediately that the black cells must be *colored* and

the dotted cells must be *blank*. The state of the other cells is dependent on the type and state of the gadget.

The gadgets are shown in Figure 7.5. All of them have $G = 7$. As it does not influence the functionality, on each side these are surrounded by only one separation line. (In the large construction we will use $D = 5$, with obvious adaptations of the descriptions.) These are already *colored*. If a cell corresponding to an input or output edge is *colored*, this means that the edge is pointing away from the vertex, and vice versa.



(a) AND (b) OR

(c) FANOUT (d) CHOICE

Figure 7.5: Nonogram gadgets.

For both the AND and the OR gadget the cells marked with a and b are corresponding to the input edges, and c is corresponding to the output edge. For both the FANOUT and CHOICE gadget the cell marked with a corresponds with the input edge, and the cells marked with b and c correspond with the output edges. It can be seen that x (with $x \in \{a, b, c\}$) is *colored* if and only if $\overline{x}$ is *blank*. We will use this property to propagate data through the gadgets. Given the values for the cells corresponding to the input edges, all gadgets (except the CHOICE gadget) are within the "simple" class, which can be easily solved. Given also the value for a cell corresponding to one of the output edges, the CHOICE gadget is also within the "simple" class. For a definition and solving algorithm of the "simple" class, the reader is referred to [1].

Lemma 7.2.1 *The construction in Figure 7.5(a) satisfies the same constraints as a Bounded NCL AND vertex, with* a *and* b *corresponding to the input edges, and* c *corresponding to the output edge.*

Proof We need to show that c can (and must) be *colored* if and only if both a and b are *blank*.

Using an algorithm as proposed in [1] we can demonstrate that there are four solutions to this Nonogram, which are shown in Figure 7.6. Figure 7.6(a) shows a solution in which both a and b are *colored*, and this results in the fact that c is *blank*. Figure 7.6(b) and Figure 7.6(c) show solutions in which either a or b is *colored*, and this also results in the fact that c is *blank*. Figure 7.6(d) shows a solution in which a and b are both *blank*. This results in the fact that a is *colored*.

Lemma 7.2.2 *The construction in Figure 7.5(b) satisfies the same constraints as a Bounded NCL OR vertex, with* a *and* b *corresponding to the input edges, and* c *corresponding to the output edge.*

Proof We need to show that c can (and must) be *colored* if and only if either a or a is *blank*, or both.

Using an algorithm as proposed in [1] we can demonstrate that there are four solutions to this Nonogram, which are shown in Figure 7.7. Figure 7.7(a) shows a solution in which both a and a are *colored*, this results in the fact that c is *blank*. Figure 7.7(b) and Figure 7.7(c) show solutions in which either a or b is *colored*, this results in the fact that c is also *colored*. Figure 7.7(d)

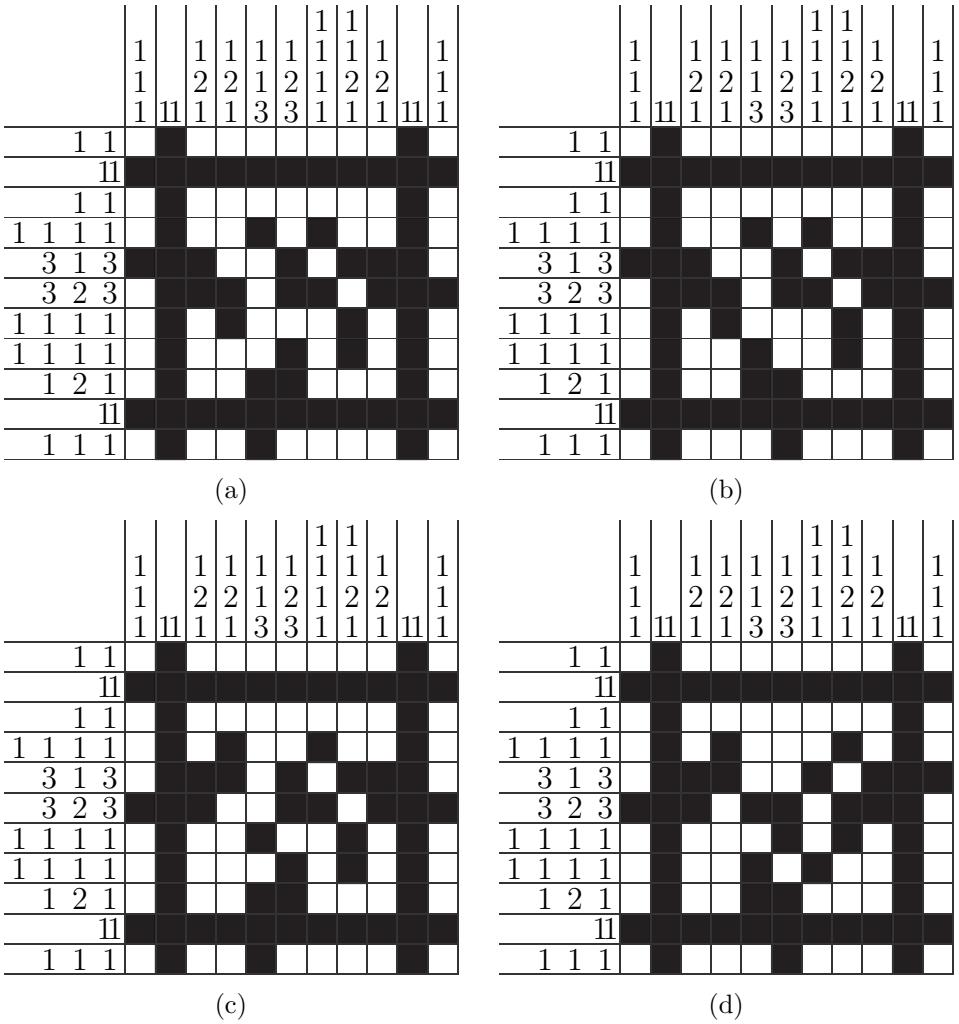


Figure 7.6: The four solutions of the AND gadget.

shows a solution in which a and b are both *blank*. This also results in the fact that c is *colored*.

Lemma 7.2.3 *The construction in Figure 7.5(c) satisfies the same constraints as a Bounded NCL FANOUT vertex, with* a *corresponding to the input edge, and cells* b *and* c *corresponding to the output edges.*

Proof We need to show that both b and c can (and must) be *colored* if and only if a is *blank*.

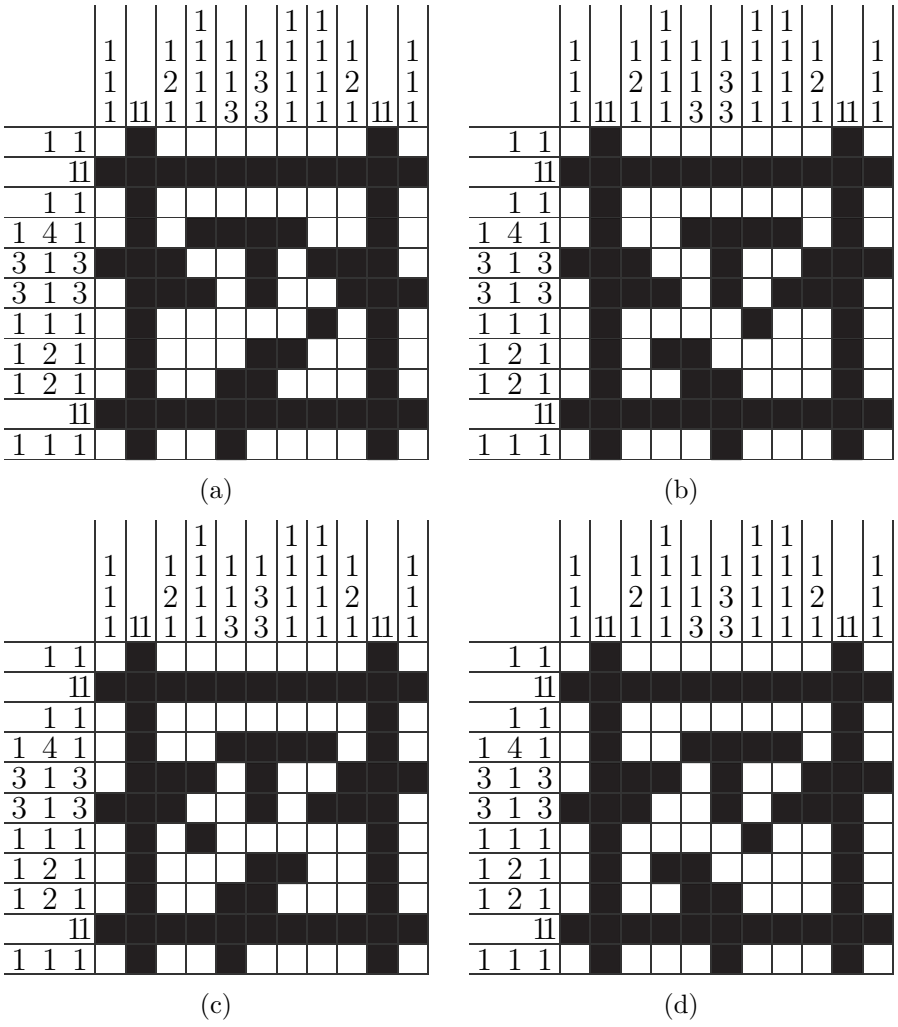
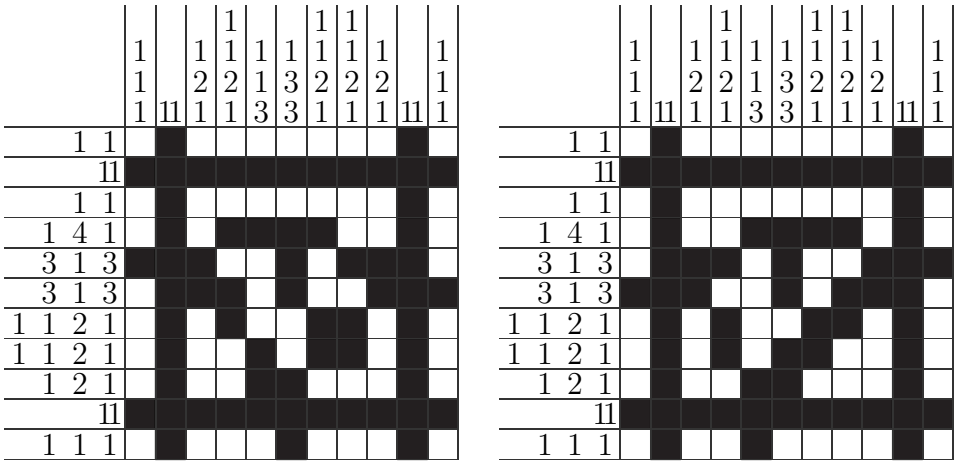


Figure 7.7: The four solutions of the OR gadget.

Using an algorithm as proposed in [1] we can demonstrate that there are two solutions to this Nonogram, which are shown in Figure 7.8. Figure 7.8(a) shows a solution in which a is *colored*, this results in both b and c being *blank*. Figure 7.8(b) shows a solution in which a is *blank*, this results in both b and c being *colored*.

Lemma 7.2.4 *The construction in Figure 7.5(d) satisfies the same constraints as a Bounded NCL CHOICE vertex, with* a *corresponding to the input edge, and cells* b *and* c *corresponding to the output edges.*



(a) (b)

Figure 7.8: The two solutions of the FANOUT gadget.

Proof We need to show that either b or c can (and must) be *colored*, if and only if a is *blank*.

Using an algorithm as proposed in [1] we can demonstrate that there are three solutions to this Nonogram, which are shown in Figure 7.9. Figure 7.9(a) shows a solution in which a is *colored*, this results in both b and c being *blank*. Figure 7.9(b) and Figure 7.9(c) show solutions in which a is *blank*. It can be seen that now either b or c can be *colored*.

Theorem 7.2.5 *Solving Nonograms is NP-complete.*

Proof We can simulate a Planar Bounded NCL graph on a Nonogram grid using the global layout of Figure 7.2 and the gadgets shown in Figure 7.5. We can use the AND gadget as wire between two gadgets. It can be used straight or as a corner. We can deploy a FANOUT gadget (as shown in Figure 7.5(c)) as a so-called victory gadget, representing the target edge. For doing so we need to adjust the description of the last column in $(5, 5)$ and of course the descriptions of all rows accordingly. This ensures that the puzzle can be solved if and only if a is *blank*, which can only be the case if the target edge in the corresponding Planar Bounded NCL graph can be flipped. The result is shown in Figure 7.10.

The authors of [4] have proven that a graph with maximal degree 3 can always be stored in a square grid of width v, where v is the number of vertices

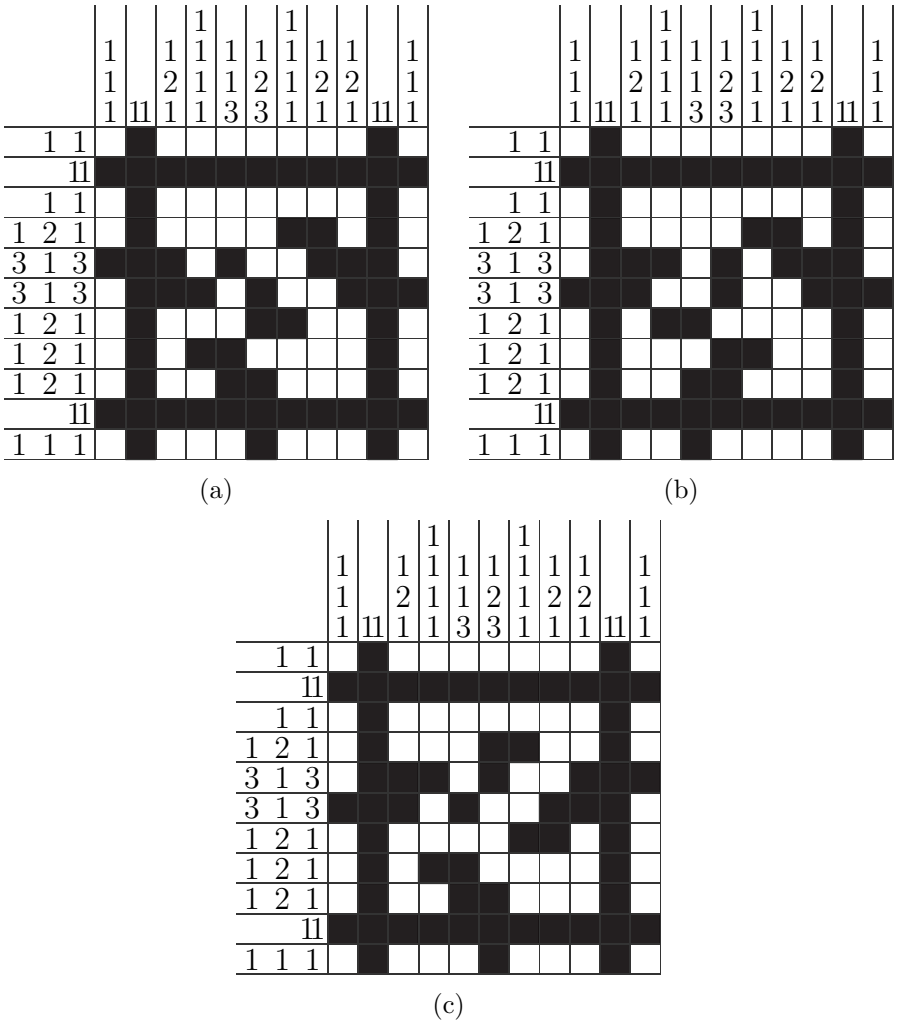


Figure 7.9: The three solutions of the CHOICE gadget.

contained by the graph. This ensures that the number of rows (and columns) used in our reduction is bounded linearly by the number of vertices in the corresponding Planar Bounded NCL graph.

An interesting problem that needs to be addressed is the fact that a solution to Nonograms is static (i.e., a binary image), while a solution to Bounded NCL is dynamic (i.e., a sequence of moves). Translating a solution to a Nonogram back to the Planar Bounded NCL graph is not trivial. For every Nonogram gadget, we can see what the final state of the Planar Bounded

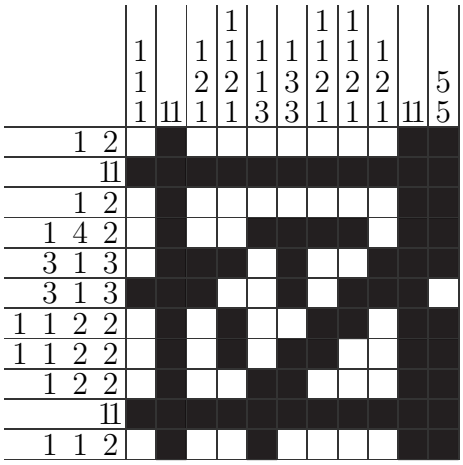


Figure 7.10: Victory gadget.

NCL vertex should be. As we have shown in Section 4.3, all instances of Planar Bounded NCL that we are interested in are acyclic. We can make a topological sort on the vertices of the Planar Bounded NCL graph, check their final state in the associated Nonogram and flip their edges, if necessary.

Nonograms are clearly in NP, as any potential solution can be verified in polynomial time.

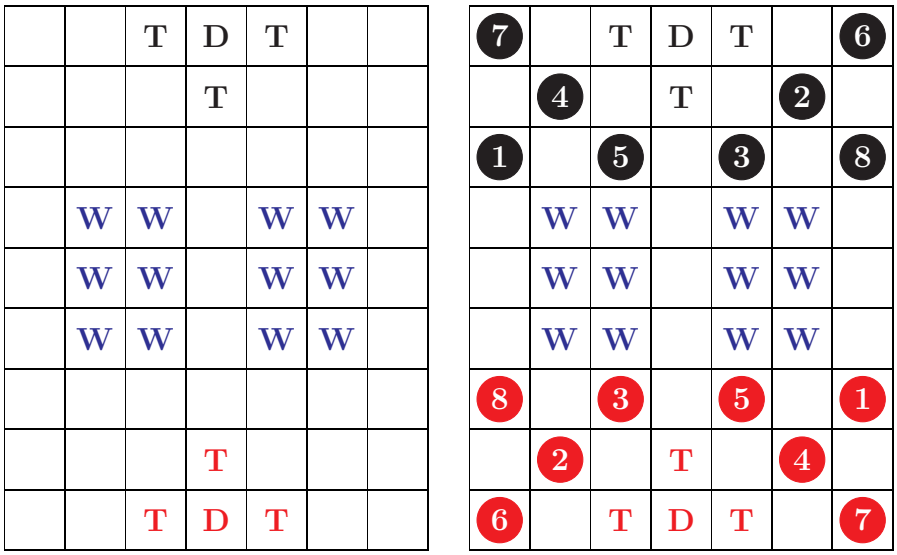
Chapter 8

Dou Shou Qi

Dou Shou Qi, also known as Jungle, Jungle Chess or Animal Chess, is a Chinese board game featuring a 9×7 square board and two players with 8 pieces each, representing wild animals. Both players are assigned a side of the board, where their pieces are positioned. The objective of the game is to move a piece onto a certain square at the opposing side of the board. The pieces represent common jungle animals, hence the western name Jungle. Although there are some similarities with western games as Chess and Stratego, little research has been done on this game. The authors of [2] have developed a heuristic for measuring assess certain position. We will give a formal definition of the game and show it PSPACE-hard.

8.1 Definition

Dou Shou Qi is played on a board consisting of 9 rows and 7 columns. Each intersection of a row and a column is a *square*. A schematic representation of the empty game board is shown Figure 8.1(a). The squares marked with a W are *water*, most pieces can not move on these squares. All other squares are *land*. The squares marked with a D are the players respective *dens*. The red player wins if he moves one of his pieces into the black den. The black player wins if he moves one of his pieces into the red den. Neither player can move their pieces into their own den. The squares marked with a T are the players respective *traps*. When a piece is in a trap of the opposing player, it becomes *vulnerable* to all opposing pieces, regardless their rank, until it moves out of the trap.



(a) Empty game board. (b) Initial position.

Figure 8.1: A schematic Dou Shou Qi game board showing the empty game board and the initial position.

Both players have 8 pieces, each piece is assigned a certain unique rank from the domain $\{1, 2, \ldots, 8\}$. A piece p with rank p_r is vulnerable to an opposing piece Q with rank q_r, if $q_r \geq p_r$. Also a piece with rank 8 is vulnerable to opposing pieces with rank 1. (And in some versions of the game, a piece with rank 1 is not vulnerable to the opposing piece with rank 8. For our complexity result, it does not matter which rule is applied.) In the initial game setup, the pieces are positioned as shown in Figure 8.1(b). A square on which no piece is positioned is called *vacant*, a square on which a piece of either side is positioned is called *occupied*. The red and black circles indicate the initial position of a piece on the board. The color of the piece indicates to which side it belongs, the number represents its rank. Players take turns doing a move, which is either a *step* or *leap*.

A step is defined to be the displacement of a piece p, to an adjacent square in one of the orthogonal directions. The square to which p is moved must be either vacant or occupied by a piece which is vulnerable to p. In the latter case a *capture* is made, the opposing piece will be removed from the board. All pieces, except those with rank 1, can only step on land squares. The pieces with rank 1 can also step on water tiles, however no captures can

be made while crossing the water-land boundary.

A leap is defined to be the displacement of a piece p with rank 6 or 7, several squares in one of the orthogonal directions. A leap can only be done if all squares between the start square and the final square are vacant and consist of water. The final square must be land and either vacant or occupied by a piece which is vulnerable to p. In the latter case a capture is made.

8.2 PSPACE-hardness

We will show Dou Shou Qi to be PSPACE-hard, by a reduction from Bounded 2CL. For this some generalizations must be made. The Bounded 2CL graph will be simulated on a $m \times n$ board, where both players have k pieces. For graphical reasons the player which plays in the original Bounded 2CL graph on the white edges, plays in our reduction with red pieces. We will refer to the players as the red player and the black player, respectively.

Whether a natural generalization of the game would imply that the k pieces all have a rank from the interval $[1, 8]$ or a distinct rank from the interval $[1, k]$ is open for debate. In our gadgets all pieces have a rank from the interval $[1, 8]$, but trivial adjustments can change this to a distinct rank from the interval $[1, k]$.

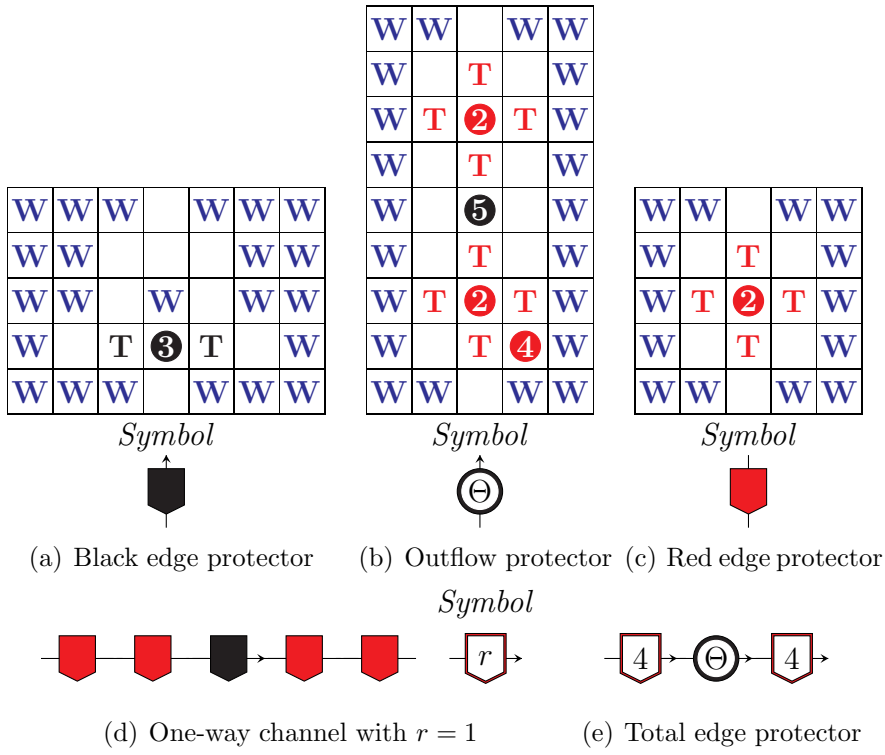
The original game board contains several properties (i.e., clustered water squares, narrow paths between the water, traps around the dens) which are symmetrical and highly regular. Which of these properties should be preserved on a generalized game board is open for debate, however in our proof we took the liberty to use water squares and traps in the gadgets. If one were to create gadgets without these squares, this would be a interesting expansion of our result.

Figure 8.2 shows the construction representing the white target edge. If the red player would be able to move a piece within this gadget, he has a straightforward way of reaching the black den and win. In the gadgets we use pieces with a rank from the interval $[2, 5]$. Note that none of the pieces will have the ability to leap, neither will we use pieces of rank 1 so no pieces can move on water squares. We will refer to pieces with rank 4 as *wolves*, as in most versions of the game this piece is represented

	T	**D**	**T**	
		T		

Figure 8.2: Red target edge.

by this animal. We will refer to pieces with rank 5
as *leopards*; pieces with rank 3 will be referred to as *hounds* and finally pieces of rank 2 will be referred to as *cats*.



(a) Black edge protector (b) Outflow protector (c) Red edge protector

(d) One-way channel with $r = 1$ (e) Total edge protector

Figure 8.3: Constructions used to support gadgets.

The reversal of an edge in the original Bounded 2CL graph will be modeled as the movement of a wolf into another gadget. Since in Bounded 2CL it is impossible to reverse an edge twice, we need a way to ensure that once moved within another gadget, the red player will not be able to move its piece back. For this we have created the constructions shown in Figure 8.3. As for the black edge protector, which is shown in Figure 8.3(a), the red player can move a wolf from bottom to top, but not the other way around. That is, unless the red player attacks from both sides. In a normal situation, the wolf would enter the gadget, the black hound will retreat behind either the left trap or the right trap, and the red wolf can pass. When passed, the black hound takes back its original position. The red wolf can not move back,

due to the traps.

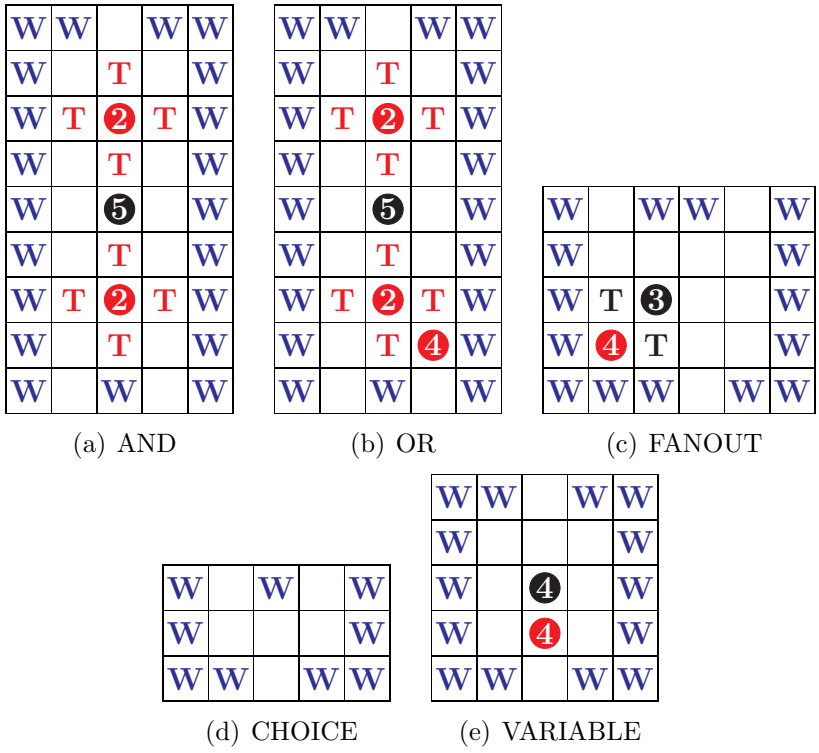
As for the red edge protector, which is shown in Figure 8.3(c), the black player can not move any piece through it, unless he attacks from both sides. He can enable one of these pieces to move through by sacrificing the other. Note that the red cat will never leave this construction in optimal play, as it is vulnerable to all other pieces.

In order to ensure that the black hound in the black edge protector will always remain within the construction, a black edge protector can be positioned between red edge protectors. A chain of four red edge protectors with a black edge protector in the middle will be called a *one-way channel*. A schematic representation of the one-way channel is shown in Figure 8.3(d). The black player can not move any piece through this construction, the red player can move any wolf from left to right. If the red player has a wolf on both sides however, he can use a simple strategy to move the wolf on the right side to the left side: The wolf on the left side will move in the black edge protector, the black hound will retreat. Instead of moving the left wolf through, the wolf on the right side can now move to the left side. We can prevent this by chaining two one-way channels to each other. Once having passed the first one way channel, the wolf which was initially on the left can still help the wolf on the right past an one-way channel; however now both pieces are on the same side they can never pass through the other one-way channel without sacrificing one of the pieces. In Figure 8.3(d), r indicates how many one-way channels are chained together. The higher the value of r is, the more pieces the red player needs to move it through this construction in the opposing direction.

In Figure 8.3(b) a *black outflow protector* is shown. This gadget is built such that the red player can move exactly one wolf through this construction, if and only if he arrives with one or two wolves at the bottom entrance of this gadget. The wolf which is already inside this gadget can never move through it without additional support; when at least one wolf arrives at the bottom of the construction, the red player can sacrifice a wolf to the black leopard. This creates space for the other wolf to move through. Note that the black player does not need to react when a piece of lower rank will be sacrificed; this piece can never reach the den. When three (or more) wolves arrive at this construction, two can move through as a result of two sacrifices. We will see further on that this will not happen. Also, the black player can never move its leopard outside the gadget; the red cats can capture it on passing a trap.

The total edge protector is shown in Figure 8.3(e). This construction contains a outflow protector between two one-way channels with $r = 4$. A total edge protector has the following properties:

- The black player can not move any piece through it, due to the red edge protectors in the one-way channel.
- The red player can only move a wolf through it in the direction of the arrow, due to the black edge protectors. In order to pass the first four black edge protectors in the opposite direction, he needs at least four wolves. We will see further on that this will never be the case.
- The red player can never benefit from moving two wolves through it, due to the outflow protector.



(a) AND (b) OR (c) FANOUT

(d) CHOICE (e) VARIABLE

Figure 8.4: Dou Shou Qi gadgets.

The main gadgets are shown in Figure 8.4. An *entrance square* is every land square adjacent to the border of a construction. Intuitively, each square

through which a piece can enter or leave the construction. Entrance squares can be of the type *input square* (entrances representing the tail of an input edge) and *output square* (entrances representing the head of an output edge). Total edge protectors will be connected to each entrance, in such a way that red wolves can only enter the gadget through the input squares, and leave the gadget through the output squares.

Lemma 8.2.1 *The construction in Figure 8.4(a) satisfies the same constraints as a Bounded 2CL AND vertex, with the entrance squares at the bottom corresponding to the tail of the input edges and the entrance square at the top to the head of the output edge.*

Proof Note that this gadget is highly similar to the black outflow protector. The red player can move a piece to the output square if and only if he arrives with two wolves at the input squares. In order to pass black's leopard, the red player needs to sacrifice one of the wolves; the other piece can pass. The total edge protectors at the input squares prevent actions from the red player that can wreck the gadget. I.e., these constructions make it impossible to leave through an entrance square.

Lemma 8.2.2 *The construction in Figure 8.4(b) satisfies the same constraints as a Bounded 2CL OR vertex, with the entrance squares at the bottom corresponding to the tail of the input edges and the entrance square at the top to the head of the output edge.*

Proof Note that this gadget is almost the same as the black outflow protector, the only difference is the additional entrance at the bottom. The red player needs at least one wolf arriving at the gadget. In order to pass the black leopard, one of the wolves must be sacrificed; another wolf can pass. Whether the red player arrives with one or two wolves does not matter, only one can pass. The total edge protectors at the input squares prevent actions from the red player that can wreck the gadget. I.e., these constructions make it impossible to leave through an entrance square.

Lemma 8.2.3 *The construction in Figure 8.4(c) satisfies the same constraints as a Bounded 2CL FANOUT vertex, with the entrance square at the bottom corresponding to the tail of the input edge and the entrance squares at the top to the head of the output edges.*

Proof The wolf can not move before the black hound moves away from the traps, otherwise it will be captured. When a wolf arrives in the gadget, the black piece can be driven away from the traps and white has two wolves, which can each leave through an output at the top. The total edge protectors at the entrances prevent actions from the red player that can wreck the gadget. I.e., these constructions make it impossible for both wolves leave through the same output square, and for any wolf to leave through the input square.

Lemma 8.2.4 *The construction in Figure 8.4(d) satisfies the same constraints as a Bounded 2CL CHOICE vertex, with the entrance square at the bottom corresponding to the tail of the input edge and the entrance squares at the top to the head of the output edges.*

Proof Once the red player moves a piece into the gadget, he can either move it through the left output or through the right output. The one-way channels connected to the outputs prevent the piece from moving back, once it has moved through one of the outputs. The total edge protectors at the output squares prevent a red wolf from moving back, once it has moved through one of the output squares.

Lemma 8.2.5 *The construction in Figure 8.4(e) satisfies the same constraints as a Bounded 2CL VARIABLE vertex, with the entrance at the top corresponding to the tail of the white output edge.*

Proof If the red player wants to set the corresponding Bounded 2CL variable to true, he can make the red wolf capture the black wolf. If the black player wants to set the corresponding variable to false, he can make the black wolf capture the red wolf. The black wolf can never move through the output at the top, due to the total edge protector.

Theorem 8.2.6 *Dou Shou Qi is PSPACE-hard.*

Proof Reduction from Bounded 2CL. Given a constraint graph made of AND, OR, FANOUT, CHOICE and VARIABLE vertices, we construct a corresponding Dou Shou Qi game board where the red player has a forced win if and only if he has a forced win on the original Bounded 2CL graph; otherwise the black player has a forced win. Note that there are no draws in bounded 2CL, neither are there in the reduction by optimal play.

The red target edge must be chained to a black edge protector. This ensures that no other red pieces except the wolves can reach the victory gadget. Black will not benefit from moving the hound into the gadget representing the target edge itself; the hound is still unable to stop a red wolf.

The red player can move a piece into the red target edge if and only if he can set the corresponding Bounded 2CL graph to true. The black player always has the potential to move a piece in the red den, but only after a certain amount of moves. If the corresponding Bounded NCL graph can be set to true, by that time the red player has already reached the black den.

We can not prove Dou Shou Qi to be PSPACE-complete, as it is probably not in PSPACE. As there is no bound on the number of moves, it is likely that Dou Shou Qi is in EXPTIME, or even in EXPSPACE.

Chapter 9

Rummikub

In this chapter we will present our partial results on Rummikub. We do not have a proof that can classify this game in any complexity class, but we hope that our initial results can help others when trying to solve this problem. The game can be played with two, three or four players, but also single player versions can be composed. We were able to construct gadgets that simulated AND, OR, CHOICE and FANOUT behaviour, but we have not yet found a way to make individual gadgets interact with each other.

We will give a formal definition of the game and show the gadgets we have created.

9.1 Definition

A Rummikub *tile set* contains m suits, numbered from 1 to m, each suit distinguished by a color. Each suit contains n tiles, numbered from 1 to n. This number represents the *value* of a tile. For each game of Rummikub, k tile sets are used together. Also J *jokers* are included. A joker is a so-called wild card, and therefore allowed to represent any other tile. In a typical game of Rummikub two tile sets are used, both consisting of four suits (orange, red, blue, black) all containing 13 tiles. Also 2 jokers are used, making a total of 106 tiles.

The *pool* is defined to be the stack of all tiles, containing them in random order. Each player is provided with 14 tiles (which are hidden to other players) and an order of taking turns is decided, that will be fixed for the entire game. Players take turns in playing tiles. A tile that is *played*, is re-

moved from the players possession, and placed open on the table. Each turn, a player can play as many tiles as he wants. If a player declines to play any tiles, he is provided with one of the tiles remaining in the pool, if it contains any. Any tile that is played must be element of either a *set* or a *row*. A set is defined to be a group of at least three tiles, all of the same value and all of a different color. A row is defined to be a group of at least three tiles, all of the same color with consecutive values. Players are also allowed to *rearrange* tiles that are already played, this makes it possible to make tiles element of a different set or row. At the end of each turn all tiles must be element of exactly one set or row. This constraint is called the *set constraint*.

At the start of the game, all players are considered to be in *quarantine*. A player in quarantine is not allowed to rearrange any tiles. He is only allowed to play tiles when their total value exceeds a certain threshold T, with T being a positive integer (typically $T = 30$). After doing so, the player is no longer in quarantine.

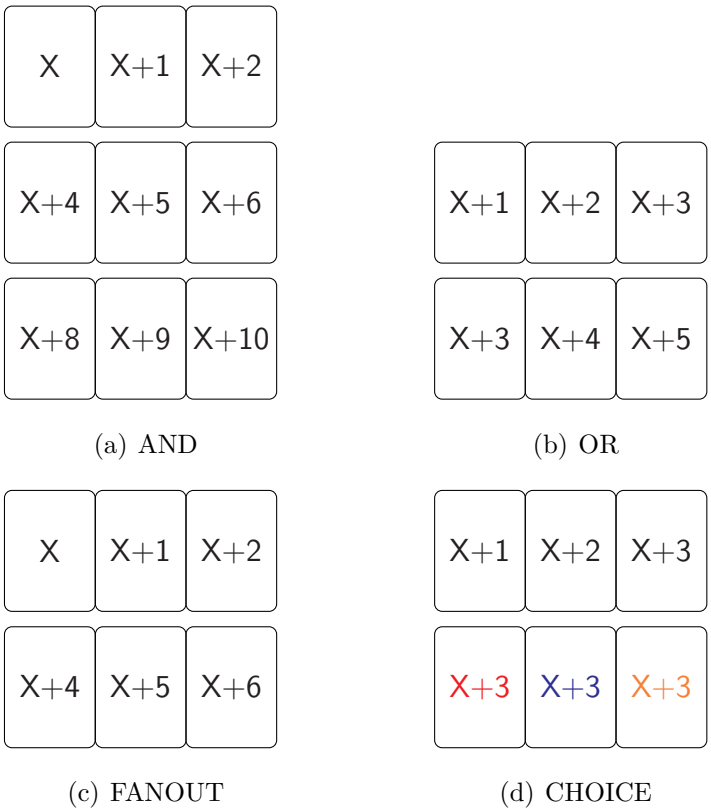
The first player who is able to play all his tiles wins the game. Whenever the pool is empty and all players decline to play tiles, the game is a draw.

9.2 Gadgets

The Rummikub gadgets are shown in Figure 9.1. Each gadget consists of 2 or 3 sets or rows and gets a range of unique values assigned to it, starting with X. The size of this range is dependent on the type of gadget. Most gadgets contain only black tiles, except for the CHOICE gadget (shown in Figure 9.1(d)), which has tile X+3 in four colors. The OR gadget (shown in Figure 9.1(b)) is the only gadget which uses the same tile twice (black X+2).

The gadgets work in such a way that a tile corresponding to an output edge can be removed from the gadget without violating the set constraint if some other tile(s) corresponding to the input edge(s) are rearranged to the gadget. We were able to create gadgets simulating the Bounded 2CL/Bounded NCL vertices, but we were unable to chain these gadgets together. Once able to rearrange a certain tile corresponding to an output edge, it is for us not yet possible to propagate this signal to other gadgets.

Lemma 9.2.1 *The construction in Figure 9.1(a) satisfies the same constraints as a Bounded NCL AND vertex, with black* X+3 *and black* X+7 *(both not in the actual gadget) corresponding to the input edges and black* X+5 *corresponding to the output edge.*



(a) AND (b) OR

(c) FANOUT (d) CHOICE

Figure 9.1: Rummikub gadgets.

Proof Black X+5 can only be rearranged from this gadget if the player is able to involve black X+3 and black X+7 in this gadget. Those can be used to rearrange these three rows in two rows, one starting at black X and ending at black X+4 and one starting at black X+6 and ending at black X+10, leaving black X+5 available.

Lemma 9.2.2 *The construction in Figure 9.1(b) satisfies the same constraints as a Bounded NCL OR vertex, with black* X *and black* X+6 *(both not in the actual gadget) corresponding to the input edges and black* X+3 *corresponding to the output edge.*

Proof One of the two black X+3 tiles can be rearranged from this gadget if the player is able to involve either black X or black X+6 in this gadget. Those can both be used to extend a row containing a black X+3 tile. A possible

problem of this gadget is that the player can obtain two black X+3 tiles, if he is able to involve both black X and black X+6 in this gadget.

Lemma 9.2.3 *The construction in Figure 9.1(c) satisfies the same constraints as a Bounded NCL FANOUT vertex, with black* X+3 *(not in the actual gadget) corresponding to the input edge and black* X *and black* X+6 *corresponding to the output edges.*

Proof Both black X and black X+6 can be rearranged if the player is able to involve black X+3 (not in the actual gadget) in this gadget. One big row starting with black X+1 and ending with black X+5 can be formed, leaving black X and black X+6 available.

Lemma 9.2.4 *The construction in Figure 9.1(d) satisfies the same constraints as a Bounded NCL CHOICE vertex, with black* X *(not in the actual gadget) corresponding to the input edge and two of the* X+3 *tiles to the output edges.*

Proof None of the X+3 tiles can be removed, without the black X tile. When the black X tile is added to the gadget, the player can choose either one of the X+3 tiles to be placed somewhere else, but not more than one.

Chapter 10

Conclusions and Future Work

We have analysed various aspect of Bounded NCL and have shown that despite some ambiguities, Planar Bounded NCL is indeed NP-complete.

We have reduced Bounded NCL to both Klondike and Mahjong and verified the prior results that these puzzles are NP-complete. We have reduced Planar Bounded NCL to Nonograms, and verified the prior result that Nonograms are NP-complete. We consider our contribution on these results amongst the verification, also the simplicity of the established proof. We have reduced Bounded 2CL to Dou Shou Qi, and proven it PSPACE-hard.

By doing so we have added weight to the claim of [6], stating that Constraint Logic is indeed a decent framework to analyse and proof the complexity of puzzles and games.

As a suggestion for future work it would be interesting to see whether Dou Shou Qi is also in PSPACE, so we can classify it as PSPACE-complete. If it turns out not to be in PSPACE, a mayor contribution would be to prove it EXPTIME-hard (for Constraint Logic can provide 2CL) or maybe even EXPSPACE-hard.

Besides the games and puzzles that are currently classified, there are games which seem to yield to a reduction from Constraint Logic, for example the Binary Puzzle and Rummikub. Reducing the appropriate Constraint Logic problem to one of these games would be an interesting result.

Bibliography

[1] Joost Batenburg and Walter Kosters. Nonograms. *Nieuwsbrief van de Nederlandse Vereniging voor Theoretische Informatica*, pages 49–62, 2012.

[2] Joseph W. Burnett. Discovering and searching loosely coupled subproblems in Dou Shou Qi. Master's thesis, Tufts University, 2010.

[3] Anne Condon, Joan Feigenbaum, Carsten Lund, and Peter Shor. Random debaters and the hardness of approximating stochastic functions. *SIAM J. Comput.*, 26:369–400, 1997.

[4] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. Algorithms for drawing graphs: An annotated bibliography. *Comput. Geom. Theory Appl.*, 4(5):235–282, 1994.

[5] David Eppstein. Computational complexity of games and puzzles. `http://www.ics.uci.edu/~eppstein/cgt/hard.html#shang`, [retrieved January 24, 2012].

[6] Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*. A K Peters, 2009.

[7] Luc Longpré and Pierre McKenzie. The complexity of Solitaire. *Theoretical Computer Science*, 410(50):5252–5260, 2009.

[8] Tadaaki Nagao, Nobuhisa Ueda, and Nobuhisa Ueda. NP-completeness results for nonogram via parsimonious reductions. Technical report, Tokyo Institute of Technology, 1996.

[9] Xiang Yan, Persi Diaconis, Paat Rusmevichientong, and Benjamin Van Roy. Solitaire: Man Versus Machine. In *Conference on Advances in Neural Information Processing*, volume 17, pages 1553–1560, 2004.