Realizations of Common Channeling Constraints in Constraint Satisfaction: Theory and Algorithms

LAM Yee Gordon

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

Abstract

Constraint satisfaction has found successes in many walks of industrial applications and computer science, such as scheduling, resource allocation, transport routing, type checking, diagram layout, among others. Typically, a problem is first modeled as a constraint satisfaction problem (CSP), which is then subject to a solver based on tree search augmented with constraint propagation algorithms.

There are usually more than one way of formulating a problem as a CSP. Channeling constraints connect and combine multiple constraint satisfaction models of the same problem to allow constraint propagation information to flow among the combined models. We identify five common channeling constraints used in the literature for connecting between integer, set, and Boolean models, and study how best to realize these channeling constraints in constraint programming systems. While the semantics of these constraints is simple, their implementations can take on various forms using the primitive constraints provided in existing solvers, such as the `iff` and the `element` constraints, thus entailing possibly different pruning behavior. There is also the possibility of global constraint implementations which enforce generalized arc consistency using specialized propagation algorithms. The thesis (1) compares the constraint propagation strengths of the different realizations of each of the five channeling constraints, which give us useful insights on proposing the best implementations of the five channeling constraints; (2) propose generic propagation algorithms for three global constraints specialized for implementing channeling. Experimentation on an extensive set of benchmark problems confirms that our proposed algorithms are in general the most efficient among all implementation possibilities.

摘要

很多工業應用和電腦科學問題，例如日程安排和工作調度、資源分配、運輸路由、類型檢查、圖佈局等，在建模為約束滿足問題後都能成功被解決。一般而言，問題首先被建模為約束滿足問題，之後以樹形搜索和限制傳播算法為基礎的解難程式來解決。

通常，每一個應用問題都可以建模為多個約束滿足問題。我們可透過雙導向約束來連接和結合同一個應用問題的多個約束滿足問題模型，從而加强約束傳播的資訊流動。我們首先認辨五種連接整數模型，集合模型，和布爾模型的雙導向約束，之後研究怎樣才能讓雙導向約束在約束編規劃系統中得到最佳的實現。儘管雙導向約束的語義簡單，但一般情況下都存在多種形式的編寫方法。假如使用約束規劃系統中預設的當且僅當約束和元素約束等，就可能導致不同的樹形搜索修剪。又如，我們可編寫全局雙導向約束，再以特殊的傳播算法强制執行以達到廣義弧形一致性。在本論文中，我們探討編寫各種雙導向約束的以不同形式，比較它們在約束傳播過程中的强度，從而洞悉實際中最佳的實施方案；第二,我們設計了三種專為全局雙導向約束的實現算法。通過廣泛的實驗，我們證實了方案算法的可行性和高效率。

Acknowledgments

I would like to thank my supervisor, Prof. Jimmy Lee, who brought me to the research area of constraint programming in 2001. Jimmy has always been a responsible supervisor and is very experienced in doing research. He is always energetic at any time. I would never forget our overnight work before paper submission deadlines. Although the results are not always rewarding.

I would also like to thank my examiners Professors Mark Wallace, Ho Fung Leung, and Philip Leong. Their constructive comments help improve the quality of the thesis a lot.

The atmosphere of our research group would not be so lively without fellow groupmates Jeff Choi, Spencer Fung, Yat Chiu Law, Clotho Tsang, Charles Siu, and May Woo. Our discussions always improved the quality of our research. Especially thanks Jeff, Spencer and Chiu give me many useful advices.

During the last two years study, I join three companies: FriarTuck (NUS, Singapore), Vocational Training Council (WanChi, Hong Kong), and Microsoft (STB, Shanghai). Each of them widens my horizon a lot both in terms of research, communication skills, personal growth industrial experience, and different culture. I am very grateful to all workmates their warm hospitality and support towards me.

Finally, I must give my best wishes to my parents and my girl friend Edith Ngai. You are always supporting me wholeheartedly throughout my four year

studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many real-life problems, such as scheduling [DSvH88], design and configuration [PS98], packing and partitioning [Hen92], combinatorial mathematics [SSW99], games and puzzles [Smi02] can be modeled as finite domain constraint satisfaction problems (CSPs) [Mac77]. The thesis reports work on a kind of constraint, channeling constraint, which is an important line of research in the constraint community, especially in redundant modeling [CCLW99]. This chapter first gives a brief introduction on constraint satisfaction problems (CSPs) and an overview of constraint solving techniques. We then introduce the concept of redundant modeling and channeling constraints, and discuss the motivations of our research. We also give an overview of the dissertation.

1.1 Constraint Satisfaction Problems

Constraint satisfaction problems (CSPs) can be defined, in the sense of Mackworth [Mac77], as follows:

> We are given a finite set of variables, a finite domain of possible values for each variable, and a conjunction of constraints. Each constraint is a relation defined over a subset of the variables, limiting the combination of values that the variables in this subset can

take. The goal is to find a consistent assignment of values from the domains to the variables so that all the constraints are satisfied simultaneously.

Solving CSPs is NP-complete [CLRS01] in general. Thus, a general solving algorithm for solving CSPs is bound to require exponential time in the worst case. A common way to solve CSPs is by backtracking tree search [GB65, Gas77, DP87, Nad89] incorporated with local consistency algorithms [Mon74, Mac77, MM88, Ger95, Ger97]. Backtracking tree search systematically explores the search space of a CSP by trying each value from the domain of each variable, and backtracking if there are any constraint violations. Local consistencies are properties, which are local to individual constraints, specifying conditions on checking whether the domains of their constrained variables are possible to be extended to a solution. Examples include node and arc consistencies [Mon74, Mac77], bounds consistency [MS98], generalized arc consistency [MM88], and set bounds consistency [Ger95, Ger97]. Local consistency algorithms enforce these properties, which cause reduction on variable domains. During backtracking tree search, removing a value from a variable domain means pruning a whole search sub-tree. Therefore, removing non-fruitful domain values effectively helps reducing the search space. Some common commercial CSP solvers such as COSYTEC CHIP [COS01], ECLiPSe [ECL05], ILOG Solver [ILO99], the CLPFD library of SICStus Prolog [SIC05], and Oz [Moz04] are based on these constraint satisfaction techniques.

1.2 Motivations and Goals

There are usually more than one way of formulating a problem into a constraint satisfaction problem (CSP). A useful modeling technique, redundant modeling [CCLW99], is to combine multiple models of the same problem using channeling constraints [CCLW99], which allow pruning information to flow among the sub-models to induce possibly further domain reduction. Various studies [FFH+02a, Smi01, LL06, HSW04] have been conducted on this topic, but different authors assume different implementations of the channeling constraints and some even do not specify how the constraints are implemented, making it difficult to compare the studies. In addition, little attention is paid to studying the best realizations of channeling constraints in existing solvers.

Channeling constraints are also constraints, and are subjected to the same treatment as other constraints in any tree search based solver augmented with local consistency algorithms. Different realizations of the channeling constraints using different underlying primitive constraints or a global constraint implementation on a certain consistency level all might entail different pruning behavior. In the thesis, we identify five common channeling constraints for connecting integer, set, and Boolean models, and enumerate how these constraints can be realized in existing solvers. We compare the constraint propagation strengths of the various realizations of each channeling constraint. We study also when and how the channeling constraint implementations can subsume some of the characteristic constraints resulting from certain model combinations. Results from this study give us useful insights and suggest the design of an efficient propagation algorithm suitable for implementing global constraints for all five channeling constraints, which is based on the notion of propagators. We propose (a) a propagation algorithm for a generalized `element` constraint for both integer and set variables specialized for implementing channeling constraints, and (b) a generic propagation algorithm for global constraint implementation of the five channeling constraints. Experimentations on an extensive set of benchmarks confirm the feasibility and efficiency of our proposed algorithms.

1.3 Outline of the Thesis

The rest of the thesis is organized as follows. Chapter 2 provides the background to the thesis. We formally define the concept of CSP, classes of variables and solutions of a CSP. We then briefly describe some CSP solving algorithms including systematic and local search solvers. In particular, we present the concept of constraint tightness [Wal01, HSW04], which is a measurement on the strength of domain reduction of constraints; and how consistency techniques can be incorporated into backtracking tree search to increase solving efficiency. Moreover, some basic graph theories are presented, which are necessary for our algorithms. Chapter 3 formally defines the concept of channeling constraints and redundant modeling. Specifically, we categorize five common types of channeling constraints, and give examples on each of them. Chapter 4 discusses the common implementation techniques of channeling constraints in existing solvers: CHIP [COS01], ECLiPSe [ECL05], SICStus Prolog [SIC05], Oz [Moz04], and ILOG Solver [ILO99]. Chapter 5 compares the constraint tightness of each type of channeling constraints among different implementations. We study also how the channeling constraints interact with the characteristic constraints arising from the particular model combinations. Chapter 6 presents our algorithms and implementations on channeling constraints. Moreover, we analyze the inefficiency of some existing channeling constraint implementations. Chapter 7 presents experimental results using our proposals. Chapter 8 presents a brief review of the related work in channeling constraints. We conclude the thesis in Chapter 9 by summarizing our contributions and giving possible directions for future research.

Chapter 2

Background

This chapter provides background to the thesis. We first give various definitions related to CSPs. Then we present constraint solving techniques for solving CSPs, which include a brief overview of systematic search and local search. In addition, we introduce the concept of constraint tightness which are used for comparing different consistency levels on constraints. Last but not least, we give some definitions on graph theory which is important for later chapters.

2.1 CSP

A *constraint satisfaction problem* (CSP) is a triple (X, D, C), where $X = \{x_1, \ldots x_n\}$ is a set of variables, $D = \{D_{x_1}, \ldots D_{x_n}\}$ is the set of domains for each variable containing the possible values for the variable, and $C = \{c_1, \ldots c_m\}$ is a set of constraints. Each constraint $c \in C$ is a relation over a subset $X_c \subseteq X$ of variables, specifying the allowed combinations of values that X_c can take.

Example 2.1. *The n-queens problem*

The n-queens problem ($Q(n)$) is to place n queens on an $n \times n$ chessboard, such that no two queens are in the same row, column, and diagonal. We formulate it as $Q_r = (X_r, D_{X_r}, C_{X_r})$ [Smi01, CLS06], where each variable $r_i \in X_r$ represents the row position of the queen in column i ($|X_r| = n$) with $D_{r_i} = \{1, \ldots, n\}$. C_{X_r} contains constraints that ensure each variable $r_i \in X_r$ must be (i) in different columns, $r_i \neq r_j$, $\forall 1 \leq i < j \leq n$; (ii) in different diagonals, $r_i + i \neq r_j + j$ and $r_i - i \neq r_j - j$, $\forall 1 \leq i < j \leq n$. Since each column must have a queen, values assigned to X_r must be a permutation of $1, \ldots, n$. ■

2.2 Classes of Variable

There are three common classes of variables, namely integer variables, Boolean variables, and integer set variables, depending on the types of values in the variables' domains. The domain of an *integer variable* [MS98, ILO99] is a set of integers. A *Boolean variable* x is a special case of an integer variable, where $D_x = \{0, 1\}$. The domain of an *integer set variable* (or simply *set variable*) [Ger94, Ger97, ILO99] is a set of integer sets.

The domain of a set variable x can be huge. When x ranges over all subsets of n possible values, $|D_x| = 2^n$. Thus, for ease of manipulation, one of the most common ways for representing the domain of a set variable x is by two sets, namely the required set and the possible set. The *required set* $RS(x)$ (or sometimes called *greatest lower bound*) of x contains all values that must belong to x, while the *possible set* $PS(x)$ (or sometimes called *least upper bound*) of x contains all values that can belong to x. Thus, $RS(x) \subseteq PS(x)$. The domain D_x of a set variable x is defined as $D_x = \{s \mid RS(x) \subseteq s \subseteq PS(x)\}$. Note that $RS(x) = \bigcap D_x$ and $PS(x) = \bigcup D_x$. A variable x is *fixed* to a value a if and only if $D_x = \{a\}$, i.e. there is only one value left in D_x. If x is a set variable, then it is the situation when $PS(x) = RS(x)$.

	r_1	r_2	r_3	r_4
1		Q		
2				Q
3	Q			
4			Q	

(a) for model Q_r

	1	2	3	4
c_1		Q		
c_2				Q
c_3	Q			
c_4			Q	

(b) for model Q_c

Figure 2.1: A solution of $Q(4)$

2.3 Solution of a CSP

By $x \mapsto a$, we denote an assignment of value $a \in D_x$ to the variable x. A *complete assignment* for a set of variables X is a set of assignments, one for each variable in X. A *solution* for a CSP (X, D, C) is a complete assignment for X satisfying all constraints in C.

Example 2.2. *The n-queens problem*

A solution $s = \{r_1 \mapsto 3, r_2 \mapsto 1, r_3 \mapsto 4, r_4 \mapsto 2\}$ of $Q(4)$ for model Q_r is shown in Figure 2.1(a). ■

An assignment $x \mapsto a$ for a variable x can be *extended* to a solution of a CSP if and only if there exists a solution s such that $(x \mapsto a) \in s$.

Example 2.3. *Suppose c is $x_1 \neq x_2$, and $D_{x_1} = \{1, 2\}$, $D_{x_2} = \{2\}$. Then $x_1 \mapsto 1$ can be extended to a solution of c, but $x_1 \mapsto 2$ cannot, since the only solution of c is $\{x_1 \mapsto 1, x_2 \mapsto 2\}$.* ■

When an integer (or Boolean) variable x is fixed to a value a, x is assigned with a automatically. Similarly, a set variable x is assigned with $RS(x)$ (or $PS(x)$) if x is fixed.

2.4 Constraint Solving Techniques

In general, CSPs are NP-complete [CLRS01]. Solving CSPs requires exponential time in terms of problem size in the worst case. There are two general classes of algorithms for solving CSPs. The first is systemic search, which explores the tree of possible assignments systematically. This can guarantee to find a solution (if it exists), or prove that no solution can be found. Thus systemic search is sound and complete. A widely used algorithm in this class is backtracking tree search [GB65, Gas77, DP87, Nad89], and it usually works with consistency techniques [Mon74, Mac77, MM88, Ger95, Ger97], which are used to remove infeasible values from variable domains so as to reduce tree size.

Another class of algorithm is stochastic local search [SLM92, DTWZ94, CLS00, ZW00], which explores the search space of complete assignments in heuristic manner. In general this may not find a solution even one exists, or prove that the problem has no solutions. Thus local search is incomplete. However, local search algorithms have been demonstrated to perform efficiently on solving some large-scale and difficult CSPs [SLM92, DTWZ94, CLS00, ZW00] when compared with algorithms based on backtracking tree search.

Our work focuses on systematic search. In the following, we describe notions and algorithms related to consistency techniques, and explain how these techniques can be incorporated into backtracking tree search.

2.4.1 Local Consistencies

There are different levels of local consistency of a constraint. In this thesis, we focus on a few common consistency levels. A constraint c is *generalized arc consistent* (GAC) [MM88] if and only if $\forall x \in X_c, \forall a \in D_x, x \mapsto a$ can be extended to a solution of c. A constraint c is *arc consistent* (AC) [Mon74, Mac77] if and only if it is *GAC* and it is binary ($|X_c|$=2). A constraint c is *set bounds consistent* (SBC) [Ger95, Ger97] if and only if $\forall x \in X_c$, $RS(x) = \bigcap S$ and $PS(x) = \bigcup S$, where $S = \{a \mid a \in D_x$ and $x \mapsto a$ can be extended to a solution of $c\}$. A constraint c is *hybrid consistent* (HC) [BHBHW05] if and only if for each integer variable $x \in X_c$, $\forall a \in D_x, x \mapsto a$ can be extended to a solution of c, and for each set variable $y \in X_c$, $RS(y) = \bigcap S$ and $PS(y) = \bigcup S$, where $S = \{a \mid a \in D_y$ and $y \mapsto a$ can be extended to a solution of $c\}$.

Typically AC and GAC are maintained for constraints containing integer (and Boolean) variables, while SBC is for constraints containing set variables only.

Example 2.4. *Suppose constraint c is $x_1 \leq x_2$, and $D_{x_1} = \{2,3,6\}$, $D_{x_2} = \{1,4,5\}$. The constraint c is not AC.*

Both $6 \in D_{x_1}$ (no value is ≥ 6 in D_{x_2}) and $1 \in D_{x_2}$ (no value is ≤ 1 in D_{x_1}) cannot be extended to any solution of c. If $D_{x_1} = \{2,3\}$ and $D_{x_2} = \{4,5\}$, then c is AC. ■

Example 2.5. *Suppose constraint c is $x_1 + x_2 = x_3$, and $D_{x_1} = D_{x_2} = \{1,2\}$, $D_{x_3} = \{1,2,3,4,5\}$. The constraint c is not GAC.*

Both 1 and 5 in D_{x_3} cannot be extended to any solution of c. If $D_{x_3} = \{2,3,4\}$, then c is GAC. ■

Example 2.6. *Suppose constraint c is $x_1 \cap x_2 = \{\}$, and $PS(x_1) = PS(x_2) = \{1,2,3\}$, and $RS(x_1) = \{2\}$, $RS(x_2) = \{1\}$. The constraint c is not SBC.*

Both $1 \in PS(x_1)$ and $2 \in PS(x_2)$ are not in any solution of c. If $PS(x_1) = \{2,3\}$ and $PS(x_2) = \{1,3\}$, then c is SBC. ■

By maintaining local consistency for each constraint, infeasible values are removed from variables' domains.

2.4.2 Constraint Tightness

Constraint tightness [Wal01, HSW04] is a kind of measurement on the strength of domain reduction of constraints with respect to different local consistencies, and we will use it for our comparing different constraint implementations. Given two sets of constraints A and B, which are defined over a same set of variables and set of domains. Φ-consistency on A is *at least as tight as* Ψ-consistency on B (written $\Phi_A \geqslant \Psi_B$) if and only if, if all constraints in A are Φ-consistent, then all constraints in B are Ψ-consistent. Φ-consistency on A is *strictly tighter then* Ψ-consistency on B (written $\Phi_A > \Psi_B$) if and only if, $\Phi_A \geqslant \Psi_B$ but not $\Psi_B \geqslant \Phi_A$. Φ-consistency on A is *as tight as* Ψ-consistency on B (written $\Phi_A = \Psi_B$) if and only if, $\Phi_A \geqslant \Psi_B$ and $\Psi_B \geqslant \Phi_A$.

Example 2.7. *Given a set of integer variables $X = \{x_1, \ldots, x_n\}$, and we want each of them to take a distinct value. We can either impose $n(n-1)/2$ pairwise disequalities ($\neq$), i.e. $x_i \neq x_j$, for $1 \leq i < j \leq n$; or use a global all-different constraint $\forall$ [Rég94] on X. We have $GAC_{\{\forall\}} > AC_{\{\neq\}}$.*

$GAC_{\{\forall\}}$ is trivially $AC_{\{\neq\}}$. Here, we give an example which is $AC_{\{\neq\}}$ but not $GAC_{\{\forall\}}$. Let $X = \{x_1, x_2, x_3\}$, and $D_{x_1} = D_{x_2} = D_{x_3} = \{1, 2\}$. There are two solutions for each pairwise disequality, while there is no solution for an all-different constraint. This is $AC_{\{\neq\}}$, but not $GAC_{\{\forall\}}$. ■

2.4.3 Tree Search

In this thesis, we assume *propagator-based* constraint solving, which is a combination of backtracking tree search and constraint propagation. This kind of search procedure features interleave of domain reduction and variable decisions. By a *variable decision* $x \rightsquigarrow v$, we mean assigning $v \in D_x$ to x (*i.e.* making $x \mapsto v$) if x is an integer or Boolean variable, and adding $v \in PS(x)$

to $RS(x)$ if x is a set variable, as well as the situation that x is fixed to a value v (i.e. $D_x = \{v\}$). By *domain reduction* $x \not\mapsto v$, we mean removing v from D_x if x is an integer or Boolean variable, and removing v from $PS(x)$ if x is a set variable. In a propagator-based solver, domain reduction is typically performed by *propagators*, each of which is attached to a constraint, for maintaining the appropriate consistency levels for the particular constraint. A propagator p is invoked whenever the domain of a variable in the constraint associated with p is changed, which can in turn prune the domains of other variables and sparkles a series of chain reaction further invoking other propagator procedures. Such a sequence of domain reduction is called *constraint propagation*, which stabilizes when all variable domains remain unchanged. The tree search procedure backtracks when (a) $D_x = \{\}$ if x is an integer or Boolean variable, or (b) $RS(x) \nsubseteq PS(x)$ if x is a set variable. Search stops or backtracks on demand when a solution is found.

Example 2.8. *$Q(4)$ is solved using propagator-based constraint solving with natural order for variable decisions. All propagators maintain AC.*

Figure 2.2 shows the resulting search tree. By symmetry, our search tree shows the branches for $r_1 \mapsto 1$ and $r_1 \mapsto 2$ only. Each chessboard represents the status of D_{X_r} after an assignment is made (e.g. $r_1 \mapsto 1$).A place in grey means no queen should be there (corresponding domain's value is reduced), where light grey means the domain's value is reduced by propagators, and dark grey means the domain's value is reduced by an assignment. An arrow means making an assignment. Since we would like to show major intermediate domain changes making by propagators, we use a dash arrow as an index of change. Now, we go though Figure 2.2 in detail.

- First, $r_1 \mapsto 1$ invokes propagators involving r_1, they are $r_1 \neq r_i$, $r_1 \neq r_i - i + 1$ and $r_1 \neq r_i + i - 1$, $\forall 2 \leq i \leq 4$. Propagators of $r_1 \neq r_i$ cause $r_i \not\mapsto 1$,

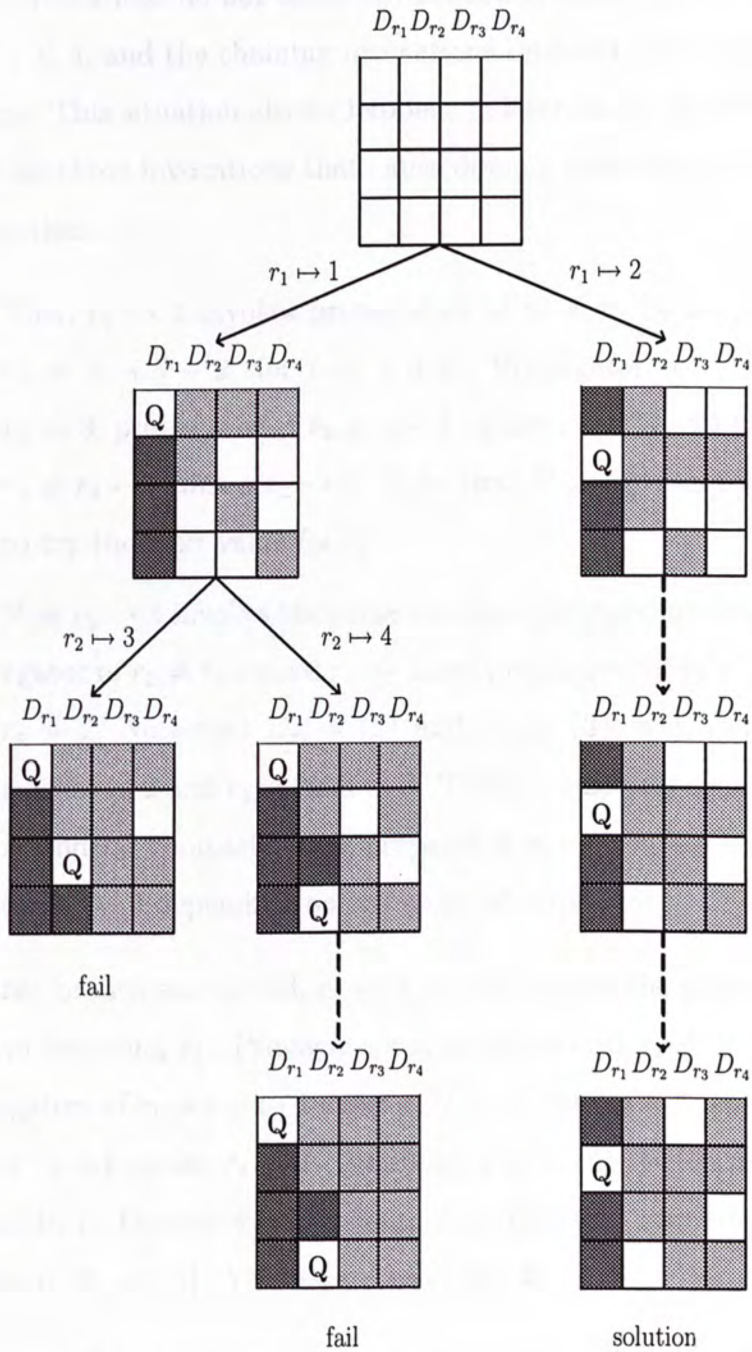


Figure 2.2: A propagator-based search tree for solving $Q(4)$

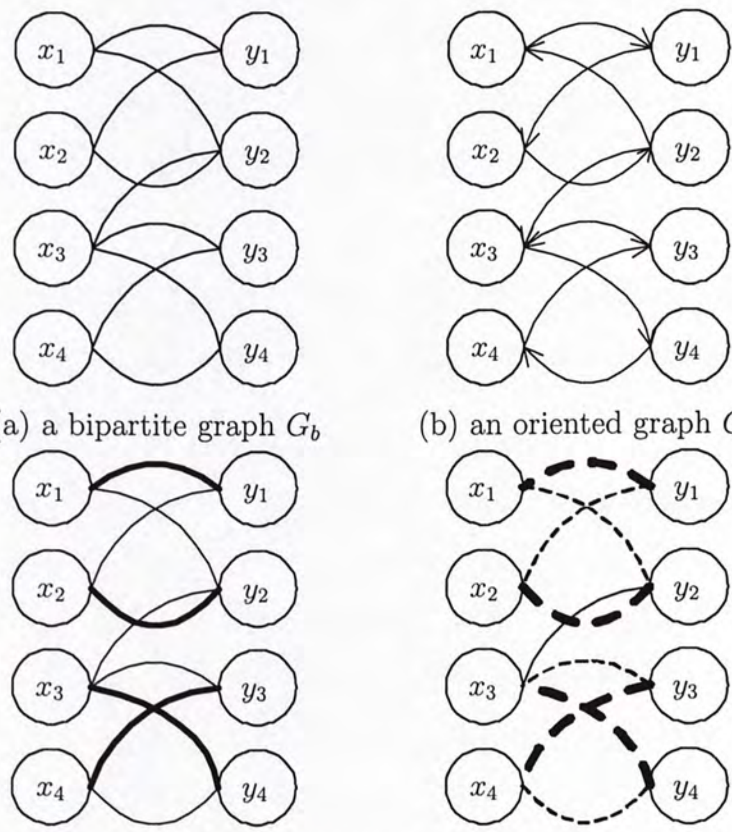
and propagators of $r_1 \neq r_i - i + 1$ cause $r_i \not\hookrightarrow i$, $\forall 2 \leq i \leq 4$. Note that some invocations do not cause any reduction effect, e.g. $r_1 \neq r_i + i - 1$, $\forall 2 \leq i \leq 4$, and the chaining invocations related to the newly domains' change. This situation always happens in later cases. Therefore, we only focus on those invocations that cause domain reduction in the following description.

- Then $r_2 \mapsto 3$ invokes propagators of $r_2 \neq r_i$, $r_2 \neq r_i - i + 2$ and $r_2 \neq r_i + i - 2$, for $i = 1, 3, 4$. Propagator of $r_2 \neq r_4$ causes $r_4 \not\hookrightarrow 3$, propagator of $r_2 \neq r_3 + 1$ causes $r_3 \not\hookrightarrow 2$, and propagator of $r_2 \neq r_3 - 1$ causes $r_3 \not\hookrightarrow 4$. Note that $D_{r_3} = \{\}$, and we backtrack to try the next value for r_2.
- Now $r_2 \mapsto 4$ invokes the same set of propagators involving r_2. Propagator of $r_2 \neq r_3$ causes $r_3 \not\hookrightarrow 4$ and propagator of $r_2 \neq r_4 + 2$ causes $r_4 \not\hookrightarrow 2$. Note that $D_{r_3} = \{2\}$ and $D_{r_4} = \{3\}$, which means that r_3 is fixed to 2 and r_4 is fixed to 3. These invoke propagators involving r_3 and r_4. Similarly, once propagator of $r_3 \neq r_4 - 1$ causes $r_3 \not\hookrightarrow 2$ (or $r_4 \not\hookrightarrow 3$ depending on the order of invocation), we backtrack.

• Another branch starts with $r_1 \mapsto 2$, which invokes the same set of propagators involving r_1. Propagators of $r_1 \neq r_i$ cause $r_i \not\hookrightarrow 2$, $\forall 2 \leq i \leq 4$, propagators of $r_1 \neq r_i - i + 1$ cause $r_i \not\hookrightarrow i+1$, for $i = 2, 3$, and propagator of $r_1 \neq r_2 + 1$ causes $r_2 \not\hookrightarrow 1$. Note that $D_{r_2} = \{4\}$, which means that r_2 is fixed to 4. This invokes propagators involving r_2. Similarly, $D_{r_3} = \{1\}$ and then $D_{r_4} = \{3\}$. We reach a solution. ■

Furthermore, by a value v being *impossible* for x, we mean $v \notin D_x$ if x is an integer or Boolean variable and $v \notin PS(x)$ is x is a set variable. By a value v being *decided* for x, we mean $x = v$ if x is an integer or Boolean variable and $v \in RS(x)$ if x is a set variable.

2.5 Graph



(a) a bipartite graph G_b (b) an oriented graph G_o

(c) a perfect matching M (d) augmenting cycles C_1 and C_2

Figure 2.3: Examples of four graph definitions

A *graph* $G = (V, E)$ consists of a set of vertices V and a set of edges E. An *edge* e is a line joining two vertices $v_i, v_j \in V$; a *directed edge* $e = (v_i, v_j)$ is an ordered pair from vertex v_i to v_j, and $e = \{v_i, v_j\}$ represents an *undirected edge.* A *directed graph* consists of only directed edges, and an *oriented graph* is a directed graph having no symmetric pair of directed edges. A *bipartite* *graph* $G = (V, E)$ consists of two disjoint sets X and Y of vertices, where $X \cup Y = V$, and $\forall e = \{v_i, v_j\} \in E$, neither $v_i, v_j \in X$ nor $v_i, v_j \in Y$. A *matching* M on a graph G is a subset of edges of G such that $\forall e_i \neq e_j \in M$, $e_i \cap e_j = \{\}$. A matching contains all vertices in G is called *perfect matching.* A *simple path* on a graph G is a sequence of distinct vertices $\{v_{i_1}, v_{i_2}, \ldots, v_{i_n}\}$ such that $\{v_{i_1}, v_{i_2}\}, \ldots, \{v_{i_{n-1}}, v_{i_n}\}$ are edges of G; a *cycle* is a path such that $v_{i_1} = v_{i_n}$. An *augmenting path* or *cycle* is a simple path or cycle whose edges are alternately in M and $E - M$, given a graph $G = (V, E)$ and a matching M.

Example 2.9. *Figure 2.3 shows (a) a bipartite graph G_b, (b) an oriented graph G_o, (c) a perfect matching M on G_b, and (d) two augmenting cycles C_1 and C_2 with respect to M and G_b.*

(a) A bipartite graph $G_b = (V, E)$ consists of two disjoint sets of vertices $X = \{x_1, \ldots, x_4\}$ and $Y = \{y1, \ldots, y_4\}$, where $V = X \cup Y$. Vertices in X and Y are connected by undirected edges. (b) An oriented graph G_o is constructed from G_b, by giving a direction for each edge in G_b. (c) A perfect matching $M = \{\{x_1, y_1\}, \{x_2, y_2\}, \{x_3, y_4\}, \{x_4, y_3\}\}$ is shown as bold edges. (d) Augmenting cycles $C_1 = \{x_1, y_1, x_2, y_2, x_1\}$ and $C_2 = \{x_3, y_4, x_4, y_3, x_3\}$ are shown as dash edges. Note that the bold dash edges are M. ■

Chapter 3

Common Channeling Constraints

In this chapter, we illustrate the relationship between models and channeling constraints. We first introduce different ways of modeling given a problem P. Then we present the concept of redundant modeling, which use channeling constraints to combine more than one model of the same problem P. Moreover, we formally define five different forms of channeling constraints. They are int-int channeling constraint, set-int channeling constraint, set-set channeling constraint, int-bool channeling constraint and set-bool channeling constraint. We give various examples on combining different models based on six problems.

3.1 Models

Given a problem P. The modeling process consists of determining the set X of variables, the corresponding domains D of variables, and the required constraints C, resulting in *model* $M = (X, D, C)$ for P. By considering P from different perspectives, we can usually find more than one way of formulating P into a CSP. Consider P as the n-queens problem in Example 2.1. We can have another model Q_c. In Model $Q_c = (X_c, D_{X_c}, C_{X_c})$ [Smi01, CLS06], the queens must be placed in different rows, and each variable $c_i \in X_c$ (integer variable) represents the column position of the queen in row i ($|X_c| = n$) with $D_{c_i} = \{1, \ldots, n\}$. C_{X_c} contains constraints that ensure each variable $c_i \in X_c$ must be (i) in different columns, $c_i \neq c_j$, $\forall 1 \leq i < j \leq n$; (ii) in different diagonals, $c_i + i \neq c_j + j$ and $c_i - i \neq c_j - j$, $\forall 1 \leq i < j \leq n$. Since each row must have a queen, values assigned to X_c must be a permutation of $\{1, \ldots, n\}$. Figure 2.1(b) gives a solution of Q_c: $\{c_1 \mapsto 2, c_2 \mapsto 4, c_3 \mapsto 1, c_4 \mapsto 3\}$. Q_c and Q_r are said to be *redundant* with respect to each other, as each of them suffices to specify the n-queens problem completely. In the next section, we illustrate how to combine different models of a problem by channeling constraints, in order to achieve additional constraint propagation. This is called *redundant modeling* [CCLW99].

Integer models, *set models* and *Boolean models* are CSP models containing only integer variables, set variables and Boolean variables respectively. We give more examples of modeling in the next section.

3.2 Channeling Constraints

Given two models M_X and M_Y of a problem with two disjoint sets of variables X and Y respectively, *channeling constraints* [CCLW99] can be used to join M_X and M_Y together by relating X and Y. There is no agreed definition of what channeling constraints should look like. Cheng et al. [CCLW99] suggest the following general form:[1]

> The variable associated with object x of type X has object y of type Y as value if and only if the variable associated with y has x as value.

[1]Pair-based models need a special form of channeling constraints, proposed by [Smi01].

For example, we can use the channeling constraint for joining X_r and X_c of the n-queens problem:

$$r_i = j \Leftrightarrow c_j = i \qquad \forall r_i \in X_r, \forall c_j \in X_c$$

From the literature, we can find the following five common forms of channeling constraints for connecting models with integer, Boolean, and set variables.

3.2.1 Int-Int Channeling Constraint (II)

Suppose X and Y are variables both from integer models. The *int-int (II)* channeling constraint has the following form:

$$x_i = j \Leftrightarrow y_j = i \qquad \forall x_i \in X \text{ and } \forall y_j \in Y$$

Example 3.1. *Langford's Problem*

This problem $L(k, n)$, "prob024" in CSPLib [GW99], is to arrange k sets of numbers from $\{1, \ldots n\}$ as a sequence of length $s = k \times n$, such that for each number $m \in \{1, \ldots, n\}$, there must be m numbers between each pair of m's (there are totally k m's). A particular instance $L(3, 9)$ [Mil99] of the Langford's problem is as follows:

> A 27-digit sequence includes the digits 1 to 9 three times each. There is just one digit between the first two 1's, and one digit between the last two 1's. There are just two digits between the first two 2's and two digits between the last two 2's, ..., and so on. Find all possible such sequences.

One solution of $L(3, 9)$ is 181915267285296475384639743. The following paragraphs give two possible integer models, L_p and L_d, for this problem.

In Model $L_p = (X_p, D_{X_p}, C_{X_p})$ [Smi01, HSW04, CLS06], each variable $p_{j_i} \in X_p$ (integer variable) represents the position of the ith copy of the number j

($|X_p| = s$). Thus $D_{p_{j_i}} = \{1, \ldots, s\}$ represents the possible positions for this number. C_{X_p} contains constraints that ensure the spacing between each pair of copies, $p_{i_{j+1}} = p_{i_j} + i + 1$, $\forall 1 \le i \le n$, $\forall 1 \le j \le k-1$. Since each number needs to take a different position, values assigned to X_p must be a permutation of $\{1, \ldots, s\}$, i.e. $p_{j_i} \neq p_{l_m}$, $\forall 1_1 \le j_i < l_m \le n_k$ in the order of $1_1, \ldots, 1_k, \ldots, n_1, \ldots, n_k$.

In Model $L_d = (X_d, D_{X_d}, C_{X_d})$ [Smi01, HSW04, CLS06], each variable $d_i \in X_d$ (integer variable) represents the number at position i ($|X_d| = s$). Thus $D_{d_i} = \{1_1, \ldots, 1_k, \ldots, n_1, \ldots, n_k\}$ represents the possible numbers at this position, where j_i denote the ith copy of the number j. C_{X_d} contains constraints that ensure the spacing between each pair of copies, $d_i = j_1 \Leftrightarrow d_{i+(m-1)(j+1)} = j_m$, $\forall 1 \le i \le s$, $\forall 1 \le j \le n$, $\forall 2 \le m \le k$, where $(i + (m-1)(j+1)) \le s$; and $d_i \neq j_1$, $\forall 1 \le j \le n$, $\forall (s - (k-1)(j+1) + 1) \le i \le s$. Since each position needs to take a different number, values assigned to X_d must be a permutation of $\{1_1, \ldots, 1_k, \ldots, n_1, \ldots, n_k\}$, i.e. $d_i \neq d_j$, $\forall 1 \le i < j \le s$.

We can combine these two models by:

$$p_i = j \Leftrightarrow d_j = i \qquad \forall p_i \in X_p, \forall d_j \in X_d$$

Example 3.2. *All Interval Series Problem*

This problem $A(n)$, "prob007" in CSPLib [GW99], is to arrange numbers from 1 to n as a sequence of length n, such that the absolute differences between every pair of neighboring numbers form the set $\{1, \ldots, n-1\}$. A solution of $A(4)$ is 1423. The following paragraphs give two possible models, A_p and A_d, for this problem.

In Model $A_p = (X_p, D_{X_p}, C_{X_p})$ [CLS06], each variable $p_i \in X_p$ (integer variable) represents the position of the number i ($|X_p| = n$). Thus $D_{p_i} = \{1, \ldots, n\}$ represents the possible positions for this number. Choi et al. [CLS06] suggest auxiliary variables $V = \{v_1, \ldots, v_{n-1}\}$ denote the position where the difference values 1 to $n-1$ belong, where $D_{v_i} = \{1, \ldots, n-1\}$, $\forall v_i \in V$. C_{X_p} contains constraints (i) relate variables in V and X_p, $(p_i - p_j = 1) \rightarrow (v_{j-i} = p_j)$ and $(p_j - p_i = 1) \rightarrow (v_{j-i} = p_i)$, $\forall 1 \leq i < j \leq n$, (ii) ensure every pair of positions for the difference value are different, $v_i \neq v_j$, $\forall 1 \leq i < j \leq n-1$. Since each number needs to take a position, values assigned to X_p must be a permutation of $\{1, \ldots, n\}$, i.e. $p_i \neq p_j$, $\forall 1 \leq i < j \leq n$. Furthermore, Choi et al. [CLS06] observe the fact that only the numbers 1 and n can give the difference of $n-1$. Thus, they suggest adding two redundant constraints $|p_1 - p_n| = 1$ and $v_{n-1} = min(p_1, p_n)$.

In Model $A_d = (X_d, D_{X_d}, C_{X_d})$ [PR01, CLS06], each variable $d_i \in X_d$ (integer variable) represents the number at position i ($|X_d| = n$). Thus $D_{d_i} = \{1, \ldots, n\}$ represents the possible numbers at this position. Choi et al. [CLS06] suggest auxiliary variables $U = \{u_1, \ldots, u_{n-1}\}$ to denote the difference between adjacent numbers, where $D_{u_i} = \{1, \ldots, n-1\}$, $\forall u_i \in U$. C_{X_d} contains constraints (i) relate variables in U and X_d, $u_i = |x_i - x_{i+1}|$, $\forall 1 \leq i \leq n-1$ (ii) ensure every differences between adjacent numbers are different, $u_i \neq u_j$, $\forall 1 \leq i < j \leq n-1$. Since each position need to take a number, values assigned to X_d must be a permutation of $\{1, \ldots, n\}$, i.e. $d_i \neq d_j$, $\forall 1 \leq i < j \leq n$.

We can channel these two models by:

$$p_i = j \Leftrightarrow d_j = i \qquad \forall p_i \in A_p, \forall d_j \in A_d$$

Moreover, we can add redundant channelling constraints between V and U as well:

$$u_i = j \Leftrightarrow v_j = i \qquad \forall u_i \in U, \forall v_j \in V$$

week \ golfer	1	2	3	4	5	6
1	1	1	2	2	3	3
2	1	2	1	3	2	3
3	1	2	2	3	3	1

(a) G_g

week \ group	1	2	3
1	{1, 2}	{3, 4}	{5, 6}
2	{1, 3}	{2, 5}	{4, 6}
3	{1, 6}	{2, 3}	{4, 5}

(b) G_p

golfer \ group	1	2	3
1	{1, 2, 3}	{}	{}
2	{1}	{2, 3}	{}
3	{2}	{1, 3}	{}
4	{}	{1}	{2, 3}
5	{}	{2}	{1, 3}
6	{3}	{}	{1, 2}

(c) G_w

week	1			2			3		
golfer \ group	1	2	3	1	2	3	1	2	3
1	1	0	0	1	0	0	1	0	0
2	1	0	0	0	1	0	0	1	0
3	0	1	0	1	0	0	0	1	0
4	0	1	0	0	0	1	0	0	1
5	0	0	1	1	0	0	1	0	0
6	0	0	1	0	0	1	1	0	0

(d) G_z

Figure 3.1: Four equivalent solutions of $G(3,2,3)$ in models G_g, G_p, G_w and G_z respectively

3.2.2 Set-Int Channeling Constraint (SI)

Suppose X are variables from a set model, and Y are variables from an integer model. The *set-int (SI)* channeling constraint has the following form:

$$j \in x_i \Leftrightarrow y_j = i \qquad \forall x_i \in X \text{ and } \forall y_j \in Y$$

Example 3.3. *Social Golfer Problem*

This problem $G(g, s, w)$, "prob010" in CSPLib [GW99], is to schedule g groups of golfers, each group has s golfers, for w weeks social play, such that each pair of golfers plays in the same group at most once. There are totally $n = g \times s$ golfers. A solution of $G(3,2,3)$ is shown in Figure 3.1. The following paragraphs give three possible models, G_g, G_p, and G_w for this problem.

In Model $G_g = (X_g, D_{X_g}, C_{X_g})$ [Smi01, LL06, CLS06], each variable $g_{i,j} \in X_g$ (integer variable) represents the group number for golfer i in week j ($|X_g| =$

$n \times w$). Thus $D_{g_{i,j}} = \{1, \dots, g\}$ represents the possible group numbers. C_{X_g} contains constraints that (i) each group must have s golfers, $|\{a \mid g_{a,j} = k, \forall 1 \leq a \leq n\}| = s$, $\forall 1 \leq j \leq w$, $\forall 1 \leq k \leq g$, and (ii) each pair of golfers plays in the same group at most once, $|\{a \mid g_{i,a} = g_{j,a}, \forall 1 \leq a \leq w\}| \leq 1$, $\forall 1 \leq i < j \leq n$.

In Model $G_p = (X_p, D_{X_p}, C_{X_p})$ [Smi01, LL06, CLS06], each variable $p_{i,j} \in X_p$ (set variable) represents the golfer for group i in week j ($|X_p| = g \times w$). Thus $PS(p_{i,j}) = \{1, \dots, n\}$ represents the possible golfer numbers. C_{X_p} contains constraints (i) the cardinality of each $p_{i,j} \in X_p$ must be equal to s, $|p_{i,j}| = s$, $p_{i,j} \in X_p$, (ii) the groups in each week do not contain the same golfer, $p_{i,k} \cap p_{j,k} = \{\}$, $\forall 1 \leq i < j \leq g$, $\forall 1 \leq k \leq w$, and (iii) each pair of golfers plays in the same group at most once, $|p_{i,k} \cap p_{j,l}| \leq 1$, $\forall 1 \leq i \leq g$, $\forall 1 \leq j \leq g$, $\forall 1 \leq k < l \leq w$.

In Model $G_w = (X_w, D_{X_w}, C_{X_w})$, each variable $w_{i,j} \in X_w$ (set variable) represents the week for golfer i at group j ($|X_w| = n \times g$), thus $PS(w_{i,j}) = \{1, \dots, g\}$ represents the possible weeks. C_{X_w} contains constraints (i) each golfer participate exactly once per week, i.e. $\bigcup_{i=1}^{g} w_{j,i} = \{1, \dots w\}$, $\forall 1 \leq j \leq n$; and $w_{i,j} \cap w_{i,k} = \{\}$, $\forall 1 \leq i \leq n$, $\forall 1 \leq j < k \leq g$, (ii) each group contains exactly s golfer, $|\{a \mid j \in w_{a,i}, \forall 1 \leq a \leq n\}| = s$, $\forall 1 \leq i \leq g$, $\forall 1 \leq j \leq w$, and (iii) each pair of golfers plays in the same group at most once, $|\{a \mid \forall i \in \{1, \dots, g\}, a \in (w_{j,i} \cap w_{k,i})\}| \leq 1$, $\forall 1 \leq j < k \leq n$.

We can combine G_p with G_g by:

$$k \in p_{i,j} \Leftrightarrow g_{k,j} = i \qquad \forall p_{i,j} \in X_p, \forall g_{k,j} \in X_g$$

We can combine G_w with G_g by:

$$k \in w_{i,j} \Leftrightarrow g_{i,k} = j \qquad \forall w_{i,j} \in X_w, \forall g_{i,k} \in X_g$$

Example 3.4. *Balanced Academic Curriculum Problem*

Period	1	2
Course	$\{1,4\}$	$\{2,3\}$

Figure 3.2: A solution of $B(2,4,3,6,1,3,\{2,3,3,4\},\{\langle 2,1\rangle\})$

This problem $B(n,m,a,b,c,d,L,R)$, "prob030" in CSPLib [GW99], is to schedule an academic curriculum by assigning n periods to m courses such that the maximum academic load for all periods is minimized. The parameters a and b are the minimum and maximum academic load for each period, c and d are the minimum and maximum number of courses for each period, $L = \{l_1,\ldots,l_m\}$ is a set of courses academic loads, and $R = \{\langle r_{1,2}, r_{1,1}\rangle,\ldots,\langle r_{p,2}, r_{p,1}\rangle\}$ is a set of prerequisite pair $\langle r_{i,2}, r_{i,1}\rangle$ such that course $r_{i,1}$ must be taken before course $r_{i,2}$. An optimal solution of $B(2,4,3,6,1,3,\ \{2,3,3,4\},\ \{\langle 2,1\rangle\})$ is shown in Figure 3.2. The following paragraphs give two possible models, B_p and B_c, for this problem.

In Model $B_c = (X_c, D_{X_c}, C_{X_c})$ [HKW02, CLS06], each variable $c_i \in X_c$ (set variable) represents the course number for period i ($|X_c| = n$). Thus $PS(c_i) = \{1,\ldots,m\}$ represents the possible course numbers. Choi et al. [CLS06] suggest two sets of auxiliary variables $W = \{w_1,\ldots,w_n\}$ and $T = \{t_1,\ldots,t_n\}$, where w_i represents the academic load at period i and t_i represents the number of courses at period i. C_{X_c} contains constraints (i) academic load for each period is bounded, $w_i = \sum_{j\in c_i} l_j$ and $a \le w_i \le b$, $\forall w_i \in W$, (ii) number of courses in each period is bounded, $t_i = |c_i|$ and $c \le t_i \le d$, $\forall t_i \in T$, (iii) each course appears once and only once, $c_i \cap c_j = \{\}$, $\forall 1 \le i < j \le n$; all courses must appear $(\sum_{i=1}^{n} t_i) = m$, and (iv) prerequisites must be satisfied, $(r_{i,1} \in c_j) \Rightarrow (r_{i,2} \notin c_k)$, $\forall \langle r_{i,2}, r_{i,1}\rangle \in R$, $\forall 1 \le k \le j \le n$.

In Model $B_p = (X_p, D_{X_p}, C_{X_p})$ [HKW02, CLS06], each variable $p_i \in X_p$ (integer variable) represents the period to which course i is assigned ($|X_p| = n$).

Thus $D_{p_i} = \{1, \ldots, n\}$ represents the possible period. Same as model B_c, two sets of auxiliary variables $W = \{w_1, \ldots, w_n\}$ and $T = \{t_1, \ldots, t_n\}$ are added. C_{X_p} contains constraints (i) academic load for each period is bounded, $w_i = \sum_{p_j \in X_p, p_j = i} l_j$ and $a \leq w_i \leq b$, $\forall w_i \in W$, (ii) number of courses in each period is bounded, $t_i = |\{p_j \mid p_j \in X_p, p_j = i\}|$ and $c \leq t_i \leq d$, $\forall t_i \in T$, and (iii) prerequisites must be satisfied, $p_{r_{i,2}} > p_{r_{i,1}}$, $\forall \langle r_{i,2}, r_{i,1} \rangle \in R$

We can use the following set-int channelling constraint to combine B_c with B_p:

$$j \in c_i \Leftrightarrow p_j = i \qquad \forall c_i \in X_c, \forall p_j \in X_p$$

3.2.3 Set-Set Channeling Constraint (SS)

Suppose X and Y are variables both from set models. The *set-set (SS)* channeling constraint has the following form:

$$j \in x_i \Leftrightarrow i \in y_j \qquad \forall x_i \in X \text{ and } \forall y_j \in Y$$

Example 3.5. *Social Golfer Problem*

We can use the following set-set channelling constraint to combine G_p with G_w:

$$g_{i,j} = k \Leftrightarrow j \in w_{k,i} \qquad \forall g_{i,j} \in X_g, \forall w_{k,i} \in X_w$$

Example 3.6. *Steiner Triple Systems Problem*

This problem $T(n)$, "prob044" in CSPLib [GW99], is to find a set of $m = n(n-1)/6$ triples, where each triple is subset of $\{1, \ldots, n\}$, and each pair of triples has at most one common integer. A solution of $T(7)$ is $\{\{1,2,3\}$, $\{1,4,5\}$, $\{1,6,7\}$, $\{2,4,6\}$, $\{2,5,7\}$, $\{3,4,7\}$, $\{3,5,6\}\}$. The following paragraphs give two possible models, S_n and S_p, for this problem.

In Model $S_d = (X_d, D_{X_d}, C_{X_d})$ [LL06], each variable $d_i \in X_d$ (set variable) represents the i-th triples ($|X_d| = m$). Thus $PS(d_i) = \{1, \ldots, n\}$ represents

Integer	Boolean
$x_i - x_j = a$	$(\sum_{k=1}^{m-a} z_{i,k} \times z_{j,k+a}) = 1$
$x_i - x_j \neq a$	$(\sum_{k=1}^{m-a} z_{i,k} \times z_{j,k+a}) = 0$
$x_i \neq x_j, \forall 1 \leq i < j \leq m$	$t_l = \sum_{k=1}^{m} z_{k,l}$ and $t_l \leq 1$, $\forall 1 \leq l \leq n$ and $(\sum_{l=1}^{n} t_l) = m$
$\lvert\{a \lvert x_a = b, \forall 1 \leq a \leq m\}\rvert$	$\sum_{i=1}^{m} z_{i,b}$

Figure 3.3: Mapping of common integer constraints to Boolean constraints in our introduced models

the possible integers that this triple can contain. C_{X_d} contains (i) each triple (integer set) contains three integers only, $|d_i| = 3$, $\forall d_i \in X_d$, and (ii) each pair of triples shares at most one common integer, $|d_i \cap d_j| \leq 1$, $\forall 1 \leq i < j \leq m$.

In Model $S_p = (X_p, D_{X_p}, C_{X_p})$ [LL06], each variable $p_i \in X_p$ (set variable) represents a set of triples that contain the integer i ($|X_p| = n$). Thus $PS(p_i) = \{1, \ldots, m\}$ represents the possible triples. C_{X_n} contains (i) each triple contains three integers only, $|\{a | i \in p_a, p_a \in X_p\}| = 3$, $\forall 1 \leq i \leq m$, and (ii) each pair of integers shares at most one common triple, $|p_i \cap p_j| \leq 1$, $\forall 1 \leq i < j \leq m$.

We can combine S_d and S_p by:

$$j \in d_i \Leftrightarrow i \in p_j \qquad \forall d_i \in X_d, \forall p_j \in X_p$$

3.2.4 Int-Bool Channeling Constraint (IB)

Suppose $X = \{x\}$ is a variable from an integer model, and Y are variables from a Boolean model. The *int-bool (IB)* channeling constraint have the following form:

$$x = i \Leftrightarrow y_i = 1 \qquad \forall y_i \in Y$$

All Boolean models in the following examples can be derived from the corresponding integer models. Figure 3.3 shows mapping of common integer constraints to Boolean constraints in our introduced integer models, where each integer variable x_i with $|D_{x_i}| = n$ correponds to a set of n Boolean variables $\{z_{i,1}, \ldots, z_{i,n}\}$. Auxiliary variables t_i are introduced whenever appropriate. Therefore, we leave out the description of constraints for the following Boolean models: The n-Queens Problem, Langford's Problem, All Interval Series Problem, Social Golfer Problem, and Balanced Academic Curriculum Problem.

The following paragraphs give five possible Boolean models, Q_z, L_z, A_z, G_z, and B_z, for each problem respectively.

In Model $Q_z = (X_z, D_{X_z}, C_{X_z})$, each variable $z_{r,c} \in X_z$ (Boolean variable) represents whether there is a queen at row r column c ($|X_z| = n^2$). The combined model with Q_r can be channeled by:

$$r_i = j \Leftrightarrow z_{j,i} = 1 \qquad \forall r_i \in X_r, \forall z_{j,i} \in X_z$$

The combined model with Q_c can be channeled by:

$$c_i = j \Leftrightarrow z_{i,j} = 1 \qquad \forall c_i \in X_c, \forall z_{i,j} \in X_z$$

In Model $L_z = (X_z, D_{X_z}, C_{X_z})$, each variable $z_{d,p} \in X_d$ (Boolean variable) represents whether number d is at position p ($|X_z| = k^2 \times n^2$). L_z can be combined with L_p by:

$$p_i = j \Leftrightarrow z_{i,j} = 1 \qquad \forall p_i \in X_p, \forall z_{i,j} \in X_z$$

L_z can be combined with L_d by:

$$d_i = j \Leftrightarrow z_{j,i} = 1 \qquad \forall d_i \in X_d, \forall z_{j,i} \in X_z$$

In Model $A_z = (X_z, D_{X_z}, C_{X_z})$, each variable $z_{p,d} \in X_d$ (boolean variable) represents whether number d is at position p ($|X_z| = n^2$). A_z can be channeled with A_p by:

$$p_i = j \Leftrightarrow z_{i,j} = 1 \qquad \forall p_i \in X_p, \forall z_{i,j} \in X_z$$

A_z can be combined with A_d by:

$$d_i = j \Leftrightarrow z_{j,i} = 1 \qquad \forall d_i \in X_d, \forall z_{j,i} \in X_z$$

In Model $G_z = (X_z, D_{X_z}, C_{X_z})$, each variable $z_{i,j,k} \in X_z$ (boolean variable) represents whether golfer i plays in group j at week k ($|X_z| = n \times g \times w$). A new model can be formed by combining G_g and G_z with:

$$g_{i,j} = k \Leftrightarrow z_{i,k,j} = 1 \qquad \forall g_{i,j} \in X_g, \forall z_{i,k,j} \in X_z$$

In Model $B_z = (X_z, D_{X_z}, C_{X_z})$, each variable $z_{c,p} \in X_z$ (boolean variable) represents whether course c is in period p ($|X_z| = n \times m$). B_p and B_z can be combined with:

$$p_i = j \Leftrightarrow z_{j,i} = 1 \qquad \forall p_i \in X_p, \forall z_{j,i} \in X_z$$

3.2.5 Set-Bool Channeling Constraint (SB)

Suppose $X = \{x\}$ is a variable from a set model, and Y are variables from a Boolean model. The *set-bool (SB)* channeling constraint have the following form:

$$i \in x \Leftrightarrow y_i = 1 \qquad \forall y_i \in Y$$

Again, the following Boolean models can be derived from the corresponding set models. Figure 3.4 shows a mapping of common set constraints to Boolean constraints in our introduced models, where each set variable x_i with $|PS(x_i)| = n$ is corresponding to a set of n Boolean variables $\{z_{i,1}, \ldots, z_{i,n}\}$. Therefore, we leave out the description part of constraint for the following Boolean models: Social Golfer Problem, Balanced Academic Curriculum Problem, and Steiner Triple Systems Problem.

Set	Boolean
$x_i = \{\}$	$(\sum_{k=1}^{m} z_{i,k}) = 0$
$x_i \cap x_j = \{\}$	$(\sum_{k=1}^{m} z_{i,k} * z_{j_k}) = 0$
$\|x_i\|$	$\sum_{j=1}^{m} z_{i,j}$
$\|\bigcap_{i=1}^{n} x_i\|$	$\sum_{k=1}^{m}((\sum_{i=1}^{n} z_{i,k}) = n)$
$\|\bigcup_{i=1}^{n} x_i\|$	$\sum_{k=1}^{m}((\sum_{i=1}^{n} z_{i,k}) \geq 1)$
$\|\{a\|b \in x_a, x_a \in X\}\|$	$\sum_{i=1}^{n} z_{i,b}$

Figure 3.4: Mapping of common set constraints to Boolean constraints in our introduced models

There are two set models, G_p and G_w, of the Social Golfer Problem. Thus G_p and G_z can be combined with:

$$k \in p_{i,j} \Leftrightarrow z_{k,i,j} = 1 \qquad \forall p_{i,j} \in X_p, \forall z_{k,i,j} \in X_z$$

G_w and G_z can be combined with:

$$k \in w_{i,j} \Leftrightarrow z_{i,j,k} = 1 \qquad \forall w_{i,j} \in X_w, \forall z_{i,j,k} \in X_z$$

The set model B_c of the Balanced Academic Curriculum Problem can be combined with its Boolean model B_z by:

$$j \in c_i \Leftrightarrow z_{j,i} = 1 \qquad \forall c_i \in X_c, \forall z_{j,i} \in X_z$$

For the Boolean model $S_z = (X_z, D_{X_z}, C_{X_z})$ of the Steiner Triple Systems Problem, each variable $z_{n,p} \in X_z$ (boolean variable) represents whether integer n is in triple p ($|X_z| = n^2(n-1)/6$). It can be combined with each of the two set models, S_n and S_p, of the Steiner Triple Systems Problem to form new models. The set-bool channeling constraints between S_d and S_z are:

$$j \in d_i \Leftrightarrow z_{j,i} = 1 \qquad \forall d_i \in X_d, \forall z_{j,i} \in X_z$$

The set-bool channeling constraints between S_p and S_z are:

$$j \in p_i \Leftrightarrow z_{i,j} = 1 \qquad \forall p_i \in X_p, \forall z_{i,j} \in X_z$$

3.2.6 Discussions

Assumptions

For the definition of II, SI and SS, we assume for each value a in the domain (or possible set) of each variable in X, there must exist a variable in Y corresponding to the value a, and *vice versa*. For example on SI, $\forall x_i \in X, \forall j \in PS(x_i), y_j \in Y$ and $\forall y_i \in Y, \forall j \in D_{y_i}, x_j \in X$. For the definition of IB and SB, we assume for each value a in the domain (or possible set) of x, there must exist a variable in Y corresponding to the value a; and for each variable y_a in Y, there must exist a corresponding value a in the domain (or possible set) of variable x. For example, in IB, $\forall i \in D_x, y_i \in Y$ and $\forall y_i \in Y, i \in D_x$.

Boolean Model via Channeling Constraint

There are two points to note. First, it is not necessary to define the *bool-bool channeling constraint* (BB), as it just makes two Boolean variables x and y equal, i.e. $x = y$. Second, one might argue that a one-variable model M_X in the definition of IB and SB is impractical. In practice, we would have a sequence of variables in $X = \{x_1, \ldots, x_n\}$ and a 2-dimensional array of Boolean variables $Y = \{y_{1,1}, \ldots, y_{1,m}, \ldots, y_{n,1}, \ldots, y_{n,m}\}$, where m is the size of the domain of each variable in X, or even a higher dimensional (like the Social Golfer Problem). Thus, the $\mathcal{IB}$ channeling constraints would usually be in the following form:

$$x_i = j \Leftrightarrow y_{i,j} = 1 \qquad \forall x_i \in X \text{ and } \forall y_{i,j} \in Y$$

We observe this form can be partition into n sets of constraints by each $x_i \in X$, and each pair of these sets *share no variables* at all. Thus, in terms of consistency level analysis and discussion on efficient implementation, our IB and SB definition are the most basic form for studying.

Previous Studies on Channeling Constraint

II is highly applied and studied [CCLW99, Smi01, FFH+02b, HSW04, CLS06]. SI is used for solving a nurse rostering problem [CCLW99] and the Balanced Academic Curriculum Problem [CLS06], and for breaking value symmetry [LL06]. SS, IB and SB can be used for breaking value symmetry [FFH+02a, LL06] as well.

Chapter 4

Realization in Existing Solvers

In this chapter, we categorize three different ways of expressing channeling constraints, namely *iff*, *ele*, *glo*. Furthermore, we discuss the common implementation techniques of these channeling constraints in existing solvers (CHIP, ECLiPSe, SICStus Prolog, Oz, and ILOG Solver), and give concrete examples on how to channel models Q_r and Q_c of the n-queen problem in these solvers. Our discussion is based on the channeling of two sets of variables, $X = \{x_1 \dots x_n\}$ and $Y = \{y_1 \dots y_m\}$ of size n and m respectively, which can be integer, set or Boolean variables.

Form	*iff*	*ele*
II	$x_i = j \Leftrightarrow y_j = i$	$x_{y_i} = i$ and $y_{x_i} = i$
SI	$j \in x_i \Leftrightarrow y_j = i$	$i \in x_{y_i}$ and $y_{x_i} = i$
SS	$j \in x_i \Leftrightarrow i \in y_j$	$i \in x_{y_i}$ and $i \in y_{x_i}$
IB	$x = i \Leftrightarrow y_i = 1$	$y_x = 1$ and $Y \Rightarrow x$
SB	$i \in x \Leftrightarrow y_i = 1$	$y_x = 1$ and $Y \Rightarrow x$

Table 4.1: Two ways of implementing channeling constraints

4.1 Implementation by if-and-only-if constraint

The most common way is to implement the channeling constraint directly according to their definitions (see the *iff* column of Table 4.1) as $n \times m$ if-and-only-if constraints. Most solvers have operators such as `#<=>` in ECLiPSe [ECL05] and SICStus Prolog's CLPFD library (SICStus CLPFD hereafter) [SIC05], `=` in Oz [Moz04], and `==` in ILOG Solver [ILO99], while some solvers, such as CHIP [COS01], need to split a single constraint into a pair of `if-then` constraints. In the following, when the context is clear, we use *iff* to refer to the $n \times m$ if-and-only-if constraints for implementing a particular channeling constraint.

4.1.1 Realization of *iff* in CHIP, ECLiPSe, and SICStus Prolog

Figure 4.4 shows the realization of models Q_r and Q_c for the n-queens problem in the corresponding solvers, but there is missing channeling constraints in line 4. The clause nQueensChannel($Rows, Cols, N$) creates two models nQueens($Rows, N$) and nQueens($Cols, N$). The clause nQueens($Rows, N$) in Figure 4.1 is implemented in SICStus Prolog, while the one in Figure 4.2 can be used by CHIP or ECLiPSe. The code in Figure 4.7 is the realization of *iff*, in which part (c) can be used by all the three solvers, part (a) is for CHIP only, and part (b) is for ECLiPSe or SICStus Prolog. We can use it by adding iff($Rows, Cols, 1$) in line 4 of Figure 4.4.

4.1.2 Realization of *iff* in Oz and ILOG Solver

Figure 4.5 and Figure 4.6 show two models Q_r and Q_c for the n-queens problem, which is implemented in Oz and ILOG Solver respectively. The corresponding missing channeling constraints in line 17 and line 15 can be filled in by the code in Figure 4.8 and Figure 4.9, which are the realizations of *iff* in Oz and ILOG Solver respectively.

```
nQueens(Rows, N):-                          ▷ Model Q_r
    length(Rows, N),                        ▷ set variables
    domain(Rows, 1, N),                     ▷ set domains
    generateDiag1(RowNDiag, Rows, 1).       ▷ generate new variables r_i − i,
                                              see Figure 4.3
    generateDiag2(RowPDiag, Rows, 1).       ▷ generate new variables r_i + i,
                                              see Figure 4.3
    all_different(Rows),                    ▷ no two queens on the same row
    all_different(RowNDiag),                ▷ no two queens on the same
    all_different(RowPDiag).                  diagonal
```

Figure 4.1: Realization of model Q_r (or Q_c) by solver SICStus Prolog

```
nQueens(Rows, N):-                          ▷ Model Q_r
    length(Rows, N),                        ▷ set variables
    Rows :: 1..N,                           ▷ set domains
    generateDiag1(RowNDiag, Rows, 1).       ▷ generate new variables r_i − i,
                                              see Figure 4.3
    generateDiag2(RowPDiag, Rows, 1).       ▷ generate new variables r_i + i,
                                              see Figure 4.3
    alldifferent(Rows),                     ▷ no two queens on the same row
    alldifferent(RowNDiag),                 ▷ no two queens on the same
    alldifferent(RowPDiag).                   diagonal
```

Figure 4.2: Realization of model Q_r (or Q_c) by solver ECLiPSe or CHIP

```
generateDiag1([], [], _).                         ▷ for generation of x_i − i
generateDiag1([D1|Ds], [X1|Xs], N):-
    D1# = X1 − N,
    N1 is N + 1,
    generateDiag1(Ds, Xs, N1).

generateDiag2([], [], _).                         ▷ for generation of x_i + i
generateDiag2([D1|Ds], [X1|Xs], N):-
    D1# = X1 + N,
    N1 is N + 1,
    generateDiag1(Ds, Xs, N1).
```

Figure 4.3: Clauses generateDiag1 and generateDiag2 in Figure 4.1, 4.2

```
nQueensChannel(Rows, Cols, N):-          ▷ channel two models together
    nQueens(Rows, N),                    ▷ Model Q_r, see Figure 4.1, 4.2
    nQueens(Cols, N),                    ▷ Model Q_c, see Figure 4.1, 4.2
    ...                                  ▷ place channeling constraints here
    labeling([ff], Rows).                ▷ label Rows by First Fail heuristic
```

Figure 4.4: Realization of channeling model Q_r and Q_c by CHIP, ECLiPSe, and SICStus Prolog

```
fun {Queens N}
   proc {$ Rows Cols}
      L1N = {MakeTuple c N}                      ▷ make a tuple with length N
      LM1N = {MakeTuple c N}                     ▷ make a tuple with length N
   in
      {For 1 N 1 proc {$ I}
         L1N.I = I LM1N.I =~ I
      end}
      {FD.tuple rqueens N 1#N Rows}
      {FD.distinct Rows}                         ▷ no two queens on the same row
      {FD.distinctOffset Rows LM1N}              ▷ no two queens on the same
      {FD.distinctOffset Rows L1N}                 diagonal
      {FD.tuple cqueens N 1#N Cols}
      {FD.distinct Cols}                         ▷ no two queens on the same column
      {FD.distinctOffset Cols LM1N}              ▷ no two queens on the same
      {FD.distinctOffset Cols L1N}                 diagonal
      ...                                        ▷ place channeling constraints here
      {FD.distribute ff Rows}                    ▷ label Rows by First Fail heuristic
   end
end
```

Figure 4.5: Realization of channeling model Q_r and Q_c by solver Oz

```
void nqueen(IlcManager& m, IlcInt n) {
    IlcIntVarArray rows(m, n, 0, n − 1),                  ▷ setup variables for model Q_r
    drow1(m, n), drow2(m, n);
    IlcIntVarArray cols(m, n, 0, n − 1),                  ▷ setup variables for model Q_c
    dcol1(m, n), dcol2(m, n);

    for (int i = 0; i < n; i++) {
        drow1[i] = rows[i] − i;                           ▷ generate r_i − i
        drow2[i] = rows[i] + i;                           ▷ generate r_i + i
        dcol1[i] = cols[i] − i;                           ▷ generate c_i − i
        dcol2[i] = cols[i] + i;                           ▷ generate c_i + i
    }

    m.add(IlcAllDiff(rows));                              ▷ no two queens on the same row
    m.add(IlcAllDiff(drow1));                             ▷ no two queens on the same
    m.add(IlcAllDiff(drow2));                               diagonal

    m.add(IlcAllDiff(cols));                              ▷ no two queens on the same column
    m.add(IlcAllDiff(dcol1));                             ▷ no two queens on the same
    m.add(IlcAllDiff(dcol2));                               diagonal

    ...                                                   ▷ place channeling constraints here

    m.add(IlcGenerate(x,                                  ▷ label Rows by First Fail heuristic
    IlcChooseMinSizeInt));
}
```

Figure 4.6: Realization of channeling model Q_r and Q_c by ILOG Solver

```
iffGenerate(_, [], _, _).
iffGenerate(Xn, [Ym|Ys], M, N):-      ▷ generate x_n = m ⇔ y_m = n,
    if Xn# = M then Ym# = N,                            ∀1 ≤ m ≤ n
    if Ym# = N then Xn# = M,
    M1 is M + 1,
    iffGenerate(Xn, Ys, M1, N).
```

(a) Implemented in CHIP

```
iffGenerate(_, [], _, _).
iffGenerate(Xn, [Ym|Ys], M, N):-      ▷ generate x_n = m ⇔ y_m = n,
    Xn# = M# <=> Ym# = N,                               ∀1 ≤ m ≤ n
    M1 is M + 1,
    iffGenerate(Xn, Ys, M1, N).
```

(b) Implemented in ECLiPSe or SICStus Prolog

```
iff([], _, _).
iff([Xn|Xs], Y, N):-                               ▷ take out x_n
    iffGenerate(Xn, Y, 1, N),          ▷ generate x_n = m ⇔ y_m = n
                                         ∀1 ≤ m ≤ n, see (a) and (b)
    N1 is N + 1,
    iff(Xs, Y, N1).
```

(c) Implemented in CHIP, ECLiPSe, or SICStus Prolog

Figure 4.7: Realization of *iff*, for channeling models Q_r and Q_c in Figure 4.4, which is applicable to CHIP, ECLiPSe, and SICStus Prolog

```
{For 1 N 1 proc {$ I}
    {For 1 N 1 proc {$ J}
        Rows.I := J = Cols.J := I    ▷ r_i = j ⇔ c_j = i
    end}
end}
```

Figure 4.8: Implemented in Oz, *iff* for channeling model Q_r and Q_c in Figure 4.5

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        m.add((rows[i] == j) == (cols[j] == i));    ▷ r_i = j ⇔ c_j = i
```

Figure 4.9: Implemented in ILOG Solver, *iff* for channeling model Q_r and Q_c in Figure 4.6

4.2 Implementations by Element Constraint

Another common technique uses the `element` constraint (see the *ele* column of Table 4.1). By x_{y_i}, we say that X are the principal variables indexed by variables in Y. An element constraint $x_{y_i} = a$, when both X and Y are sets of integer variables, has an equivalent meaning as:

$$y_i = j \Rightarrow x_j = a \qquad \forall j \in D_{y_i}$$

An element constraint $x_{y_i} = a$, when both X is a set of integer variable and Y is a set of set variables, has an equivalent meaning as:

$$j \in y_i \Rightarrow x_j = a \qquad \forall j \in PS(y_i)$$

An element constraint $a \in x_{y_i}$, when both X is a set of set variable and Y is a set of integer variables, has an equivalent meaning as:

$$y_i = j \Rightarrow a \in x_j \qquad \forall j \in D_(y_i)$$

An element constraint $a \in x_{y_i}$, when both X and Y are sets of set variables, has an equivalent meaning as:

$$j \in y_i \Rightarrow a \in x_j \qquad \forall j \in PS(y_i)$$

For II, SI, and SS, there are two set of element constraints, one using X and the other using Y as the principal variables. When X are the principal variables, we refer to the m constraints as ele_X:

$$x_{y_i} = i, \forall y_i \in Y, \text{for cases when } X \text{ is a set of integer variables}$$

$$i \in x_{y_i}, \forall y_i \in Y, \text{for cases when } X \text{ is a set of set variables}$$

Similarly, when Y are the principal variables, we refer to the n constraints as ele_Y:

$$y_{x_j} = j, \forall x_j \in X, \text{for cases when } Y \text{ is a set of integer variables}$$

$$j \in y_{x_j}, \forall x_j \in X, \text{for cases when } Y \text{ is a set of set variables}$$

Thus ele_X and ele_Y together are equaivalent as *iff*. For IB and SB, since ele_X can not be realized, we need *Boolean mapping constraint* $Y \Rightarrow x$:

$$y_i = 1 \Rightarrow x = i, \forall y_i \in Y, \text{for cases when } X \text{ is a set of integer variables}$$

$$y_i = 1 \Rightarrow i \in x, \forall y_i \in Y, \text{for cases when } X \text{ is a set of set variables}$$

To the best of our knowledge, existing solvers support the `element` constraint for integer variables only. CHIP [COS01], ECLiPSe [ECL05], Oz [Moz04], and SICStus CLPFD [SIC05] has an element constraint in form of `element(`*Index*`,` *List*`,` *Value*`)`, where *Index* and *Value* can be an integer or integer variable, and *List* can be a list of integers or integer variables. The meaning of the constraint is that the *Index*-th element in *List* is *Value*. ILOG Solver [ILO99] supports a syntax very close to our notation. For example, the constraints in

ele_X can be directly written as `x[y[i]] == i`. In the next section, we propose a generic propagator for a generalized `element` constraint for both integer and set variables specialized for implementing channeling constraints.

4.2.1 Realization of *ele* in CHIP, ECLiPSe, and SICStus Prolog

Figure 4.10 shows the realization of *ele* for CHIP, ECLiPSe or SICStus Prolog. We can fill in elementGenerate($Rows$, $Cols$, 1) (i.e. ele_{Rows}) and elementGenerate($Cols$, $Rows$, 1) (i.e. ele_{Cols}) in line 4 of Figure 4.4 for the missing channeling constraints.

4.2.2 Realization of *ele* in Oz and ILOG Solver

Figure 4.11 and Figure 4.12 show the realization of *ele* for Oz and ILOG Solver respectively. We can fill them correspondingly into line 17 and line 15 of Figure 4.5 and Figure 4.6 for the missing channeling constraints.

elementGenerate($\_$, [], $\_$).
elementGenerate(X, $[Yn|Ys]$, N):- $\triangleright$ take out y_n
  element(Yn, X, N), $\triangleright$ $x_{y_n} = n$
  $N1$ is $N + 1$,
  elementGenerate($X, Ys, N1$).

Figure 4.10: Code for generating *ele* for channeling models Q_r and Q_c in Figure 4.4

```
{For 1 N 1 proc {$ I}
    {FD.element Cols.I Rows I}   ▷ r_{c_i} = i
    {FD.element Rows.I Cols I}   ▷ c_{r_i} = i
end}
```

Figure 4.11: Code for generating *ele* for channeling models Q_r and Q_c in Figure 4.5

```
for (int i = 0; i < n; i++) {
    m.add(rows[cols[i]] == i);   ▷ r_{c_i} = i
    m.add(cols[rows[i]] == i);   ▷ c_{r_i} = i
}
```

Figure 4.12: Code for generating *ele* for channeling models Q_r and Q_c in Figure 4.6

4.3 Global Constraint Implementations

Last but not least, it is also possible to implement each channeling constraint as a single global constraint *glo* by designing specialized propagation algorithms to enforce consistency. As far as we know, only implementation for integer variables is supported in existing solvers, such as `inverse`, `IlcInverse`, and `assignment` in CHIP [COS01], ILOG Solver [ILO99] and SICStus CLPFD [SIC05] respectively. Note that `IlcInverse` does not enforce GAC, while `assignment` has an argument to control the consistency level. Again, we will propose another generic propagator for implementing *glo* that enforces AC on *iff* for all five channeling constraints.

4.3.1 Realization of *glo* in CHIP, SICStus Prolog, and ILOG Solver

The missing channeling constraints in line 4 of Figure 4.4 and line 15 of Figure 4.6, can be filled by inverse($Rows, Cols$) in CHIP or assignment($Rows, Cols$) in SICStus Prolog, and m.add(IlcMyInverse($rows, cols$)) in ILOG Solver respectively.

In the rest of the thesis, we focus on ILOG Solver implementations.

Chapter 5

Consistency Levels of Channeling Constraints

In this chapter, we compare the constraint tightness of each channeling constraint among different implementations. Where applicable, we study also how the channeling constraints interact with the characteristic constraints arising from the particular model combinations. For example, II is possible only for permutation problems [Smi01, Wal01, HSW04], and this enforces the characteristic constraint that all variables are different. Our major theorems show that except for II, maintaining a higher level of consistency on the entire channeling constraint does not increase the pruning power. We present these in five sections, which corresponds to II, SI, SS, IB, and SB.

In the rest of this section, we are channeling two models M_X and M_Y with variables $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$ respectively. We denote $S_{X,Y} = X \cup Y$. The following property is useful in subsequent presentations.

Property 5.1. *Given a set of constraints A, B, and C, and an Φ-consistency which can be GAC, SBC and HC:*

1. *monotonicity* **[Wal01, HSW04]***: $\Phi_{A \cup B} \geqslant \Phi_A$*
2. *fixed-point* **[Wal01, HSW04]***: If $\Phi_A = \Phi_B$, then $\Phi_{A \cup C} = \Phi_{B \cup C}$*

3. *transitivity: If* Φ_A=Φ_B *and* Φ_B=Φ_C *, then* Φ_A=Φ_C

4. *subsumption: If* $\Phi_A>\Phi_B$*, then* $\Phi_{A \cup C}=\Phi_{A \cup B \cup C}$

Proof. Point 3 is by definition. To pove point 4, $\Phi_A>\Phi_B$ means $\Phi_A \geqslant \Phi_{A \cup B}$. And by monotonicity, $\Phi_{A \cup B} \geqslant \Phi_A$, we have $\Phi_A = \Phi_{A \cup B}$. Then by fixed-point, we get $\Phi_{A \cup C} = \Phi_{A \cup B \cup C}$. □

The following lemma is useful in proving theorems concerning SBC.

Lemma 5.1. *Given a constraint c. If both* $x \mapsto RS(x)$ *and* $x \mapsto PS(x)$ *can be extended to a solution of c,* $\forall x \in X_c$*, then c is SBC.*

Proof. For each $x \in X_c$, let $S = \{a \mid a \in D_x$ and $x \mapsto a$ can be extended to a solution of $c\}$. Thus, $RS(x) \in S$, $PS(X) \in S$. Recall the property of $RS(x) \subseteq D_x \subseteq PS(x)$, we have $\forall a \in S$, $RS(x) \subseteq a \subseteq PS(x)$. Consequently, $\bigcap S = RS(x)$ and $\bigcup S = PS(x)$, and c is SBC. □

The following corollary of Lemma 5.1, which is useful in proving theorems concerning HC.

Corollary 5.2. *Given a constraint c. If for each integer variable* $x \in X_c$*,* $\forall a \in D_x, x \mapsto a$ *can be extended to a solution of c, and for each set variable* $y \in X_c$*, both* $y \mapsto RS(y)$ *and* $y \mapsto PS(y)$ *can be extended to a solution of c, then c is HC.*

5.1 Int-Int Channeling (II)

Both M_X and M_Y are integer models. Since each variable must take exactly one value, the II channeling constraint implies the following: (1) variables in X take on different values, (2) variables in Y take on different values, and (3) $m = n = |D_{x_i}| = |D_{y_j}|$ for all $i, j \in \{1, \ldots, n\}$. The characteristic constraints are thus all-different on X and the same on Y. Therefore, both M_X and M_Y are permutation problems [Smi01, Wal01, HSW04].

There are two ways to implement all-different: by a series of pairwise disequalities ($\neq$) and by a single global `allDiff` constraint ($\forall$). In the rest of the paper, we use the notation "$\{cx, cc, cy\}$" to denote the set of constraints in which cx are the characteristic constraints on X, cc is the channeling constraint implementation, and cy are the characteristic constraints on Y. For example, $\{\forall, ii, \forall\}$ means a global `allDiff` on X plus a global implementation of II on $S_{X,Y}$ and a global `allDiff` on Y. Note that cx and cy can be empty under appropriate context.

We first prove that ii w.r.t. GAC subsumes global `allDiff` constraint on both models.

Theorem 5.3. $GAC_{\{ii\}}{=}GAC_{\{\forall,ii,\forall\}}$.

Proof. By Property 5.1.1, $GAC_{\{\forall,ii,\forall\}}{\geqslant}GAC_{\{ii\}}$. To show the reverse by contradiction, suppose $S_{X,Y}$ is $GAC_{\{ii\}}$ but not $GAC_{\{\forall,ii,\forall\}}$ due to global `allDiff` constraints. W.L.O.G., assume it is not $GAC_{\{\forall\}}$ w.r.t. X (a symmetric proof can be made on Y). Then $\exists$ a value in the domain of x_i, say d_i, cannot be extended to any solution of the global `allDiff` constraint on X, but $\exists$ a solution e of ii which contains $x_i \mapsto d_i$. Hence $\exists\ k_1 \neq k_2, k$ such that $x_{k_1}{\mapsto}k$ and $x_{k_2}{\mapsto}k$ are in e. However y_k needs to take values k_1 and k_2 by the definition of ii, this is a contradiction. $\square$

Corollary 5.4. *$GAC_{\{ii\}}{=}GAC_{\{a,ii,b\}}$, where a and b can be $\forall$ or $\neq$ or empty.*

Proof. We first prove the case of a is $\neq$ and b is empty. By Theorem 5.3, we have $GAC_{\{ii\}}{=}GAC_{\{\forall,ii,\forall\}}$, and by Property 5.1.2, we get $GAC_{\{\neq,ii\}}{=}GAC_{\{\neq,\forall,ii,\forall\}}$. By Property 5.1.4 and the fact that $GAC_{\{\forall\}} > AC_{\{\neq\}}$[1], we have $GAC_{\{\neq,\forall,ii,\forall\}}{=}$

[1]Note that $GAC_{\{\neq\}}$ and $AC_{\{\neq\}}$ are equivalent.

$GAC_{\{\forall,ii,\forall\}}$. Thus, by Property 5.1.3, $GAC_{\{ii\}}= GAC_{\{\neq,ii\}}$. Similar proofs can be applied to all the other cases. □

Theorem 5.3 and Corollary 5.4 suggest that all-different (either as global `allDiff` or pairwise disequalities) do not increase the amount of overall domain reduction when ii is maintaining GAC.

Theorem 5.5. $AC_{\{iff\}}$=$GAC_{\{ele\}}$.

Proof. First, we show $GAC_{\{ele\}} \geqslant AC_{\{iff\}}$. Suppose it is $GAC_{\{ele\}}$ but not $AC_{\{iff\}}$. Consider the following two cases: (1) $\exists$ a value a in the domain of x_i which makes a constraint c, $x_i = j \Leftrightarrow y_j = i$ to be not AC (2) $\exists$ a value a in the domain of y_i which makes a constraint c, $y_i = j \Leftrightarrow x_j = i$ to be not AC. (1) Since it is $GAC_{\{ele\}}$, $a \in D_{x_i}$ implies $i \in D_{y_a}$ by ele_Y. If $a = j$, then c must be AC. If $a \neq j$, we want to show that $\exists\ b \neq i$, such that $b \in y_j$, in order to make c is AC. Suppose b does not exist, then y_j must equal i. By ele_X, x_i must equal j, which contradicts to $a \neq j$. Thus, c is AC, which is a contradiction. (2) Symmetric proof can be made as (1).

Second, we show $AC_{\{iff\}} \geqslant GAC_{\{ele\}}$. Suppose it is $AC_{\{iff\}}$ but not $GAC_{\{ele_X\}}$. Consider the following four cases: (1) $\exists$ a value j in the domain of y_i which makes a constraint c, $x_{y_i} = i$ from ele_X, to be not GAC. (2) $\exists$ a value j in the domain of y_i which makes a constraint c, $y_{x_a} = a$ from ele_Y, to be not GAC. (3) $\exists$ a value j in the domain of x_i which makes a constraint c, $y_{x_i} = i$ from ele_Y, to be not GAC. (4) $\exists$ a value j in the domain of x_i which makes a constraint c, $x_{y_a} = a$ from ele_X, to be not GAC. (1) Now we construct a complete assignment e of c. First we make $e = \{y_i \mapsto j, x_j \mapsto i\}$. Then for each $x_k \in X$ ($x_k \neq x_j$), make $e = e\ \cup\ \{x_k \mapsto d_k\}$, where $d_k \in D_{x_k}$. Thus e is a solution of c, and this is a contradiction. (2) Now we construct a complete assignemtn e of c. First we make $e = \{y_i \mapsto j\}$. We want to show $\exists\ b \in x_a$, in which $b \neq i$. Suppose b must be i, then x_a must be i, and y_i must be a by $x_a = i \Leftrightarrow y_i = a$.

This condicts with $j \in D_{y_i}$. Thus, we can make $e = e \cup \{x_a \mapsto b, y_b \mapsto a\}$. And for the rest of $y_k \neq y_b \neq y_i$, make $e = e \cup \{y_k \mapsto d_k\}$, where $d_k \in D_{y_k}$. Thus e is a solution of c, and this is a contradiction. (3) Symmetric proof can be made as (1). (4) Symmetric proof can be made as (2). □

From Theorem 5.5, we know that each constraints in *ele* w.r.t. GAC is as tight as each constraints in *iff* w.r.t. AC. In the next two theorems, we prove two tightness relations between *ii* w.r.t. GAC and *iff* w.r.t. AC.

Theorem 5.6. $GAC_{\{ii\}}{>}AC_{\{iff\}}$.

Proof. $GAC_{\{ii\}}$ is trivially $AC_{\{iff\}}$. Now we give an example which is $AC_{\{iff\}}$ but not $GAC_{\{ii\}}$. Let $X = \{x_1, \ldots, x_4\}, Y = \{y_1, \ldots, y_4\}$, and $D_{x_1} = D_{x_2} = \{1,2\}$, $D_{x_3} = D_{x_4} = D_{y_1} = D_{y_2} = \{1,2,3,4\}$, $D_{y_3} = D_{y_4} = \{3,4\}$. This is $AC_{\{iff\}}$. But $y_1 \mapsto 3$, $y_1 \mapsto 4$, $y_2 \mapsto 3$, $y_2 \mapsto 4$, $x_3 \mapsto 1$, $x_3 \mapsto 2$, $x_4 \mapsto 1$ and $x_4 \mapsto 2$ cannot be extended to any solution of ii. This is not $GAC_{\{ii\}}$. □

From Theorem 5.6, we know that *ii* w.r.t. GAC is tighter than *iff* w.r.t. AC.

Theorem 5.7. $GAC_{\{ii\}}{=}GAC_{\{\forall,iff\}}{=}GAC_{\{iff,\forall\}}$.[2]

Proof. By symmetry, we prove $GAC_{\{ii\}}{=}\ GAC_{\{\forall,iff\}}$ only. First, we show $GAC_{\{ii\}}{\geqslant}\ GAC_{\{\forall,iff\}}$. By Theorem 5.6, we have $GAC_{\{ii\}}{>}\ AC_{\{iff\}}$. Therefore, $GAC_{\{\forall,ii\}}{\geqslant}\ AC_{\{\forall,iff\}}$. By Corollary 5.4 and Property 5.1.3, we get $GAC_{\{ii\}}{\geqslant}\ GAC_{\{\forall,iff\}}$. To show the reverse by contradiction, suppose it is $GAC_{\{\forall,iff\}}$ but not $GAC_{\{ii\}}$. Then $\exists$ a value in the domain of x_i, say d_i, cannot be extended to any solution of ii, but there exists a solution $e_x = \{x_1{\mapsto}d_1, \ldots, x_i{\mapsto}d_i, \ldots, x_n{\mapsto}d_n\}$ of the global `allDiff` constraint on X. Now for each $x_j{\mapsto}d_j \in e_x$, there must exist $j \in D_{y_{d_j}}$ because of $AC_{\{iff\}}$, and we construct $e_y = \{y_{d_1}{\mapsto}1,$

[2] *Note that* $GAC_{\{iff\}}$ *and* $AC_{\{iff\}}$ *are equivalent.*

$\ldots, y_{d_n} \mapsto n\}$. Note that $\{d_1, \ldots, d_n\} = \{1, \ldots, n\}$, thus $e = e_x \cup e_y$ is a solution of *ii*. This is a contradiction. □

Corollary 5.8. *$GAC_{\{ii\}}$=$GAC_{\{a,c,b\}}$, where c can be iff, ele_X or ele_Y; a and b can be $\forall$ or $\neq$ or empty, but with a condition that at least one of a and b must be $\forall$.*

Proof. We first prove the case of $c = \textit{iff}$, $a = \forall$ and b is $\neq$, and other cases of a and b can be proved similarly. By Theorem 5.7, we have $GAC_{\{ii\}}= GAC_{\{\forall,\textit{iff}\}}$, and by Property 5.1.2, we get $GAC_{\{ii,\neq\}}= GAC_{\{\forall,\textit{iff},\neq\}}$. By Corollary 5.4 and Property 5.1.3, we have $GAC_{\{ii\}}$=$GAC_{\{\forall,\textit{iff},\neq\}}$. For $c = ele_X$ and $c = ele_Y$, by Theorem 5.5, we have $AC_{\{\textit{iff}\}}= GAC_{\{ele_X\}}= GAC_{\{ele_Y\}}$, and by Property 5.1.2, we get $AC_{\{\forall,\textit{iff}\}}= GAC_{\{\forall,ele_X\}}= GAC_{\{\forall ele_Y\}}$ and $AC_{\{\textit{iff},\forall\}}= GAC_{\{ele_X,\forall\}}=$ $GAC_{\{ele_Y,\forall\}}$. Then, similar proofs can be made for all the other cases. □

Corollary 5.8 shows that *iff* or ele_X or ele_Y plus a global `allDiff` constraint on either X or Y can achieve the same domain reduction as *ii* w.r.t. GAC.

Theorem 5.9. **[Wal01, HSW04]** $AC_{\{\textit{iff}\}}$=$AC_{\{\neq,\textit{iff},\neq\}}$.

Corollary 5.10. *$GAC_{\{c1\}}$=$GAC_{\{a,c2,b\}}$, where c1 and c2 can be iff or ele; a and b can be $\neq$ or empty.*

Proof. The cases of $c1 = c2 = \textit{iff}$ is proved by Walsh and Hnich et al. [Wal01, HSW04]. We first prove the case of $c1 = \textit{iff}$, $c2 = ele$, a is $\neq$ and b is empty. By Theorem 5.5, $AC_{\{\textit{iff}\}}$=$GAC_{\{ele\}}$. By Property 5.1.2, we have $AC_{\{\neq,\textit{iff}\}}$=$GAC_{\{\neq,ele\}}$. Thus by Property 5.1.3 and $AC_{\{\textit{iff}\}}$=$AC_{\{\neq,\textit{iff}\}}$ [Wal01, HSW04], we have $AC_{\{\textit{iff}\}}= GAC_{\{\neq,ele\}}$. Similar proofs can be made for all the other cases. □

Corollary 5.10 shows that disequalities on X and Y can be removed when AC is maintained on *iff* or GAC is maintained on *ele*.

5.2 Set-Int Channeling (SI)

We assume that M_X is a set model and M_Y is an integer model. X and Y must satisfy the characteristic condition for the channeling to make sense: (1) $\bigcup_{i=1}^{n} x_i = \{1, \ldots, m\}$ and (2) $x_i \cap x_j = \{\}$ for all $i, j \in \{1, \ldots, n\}$ and $i \neq j$. In other words, each index for variables in Y must be in exactly one set variable in X, since each variable in Y must take exactly one value. We call (1) and (2) in totality the *partition constraint*.

Again, there are two ways to implement the partition constraints: by implementing conditions (1) and (2) directly ($\|$) and by implementing a single global constraint ($\prod$) which is available in ILOG Solver [ILO99].

The following property is useful for our subsequent proofs.

Property 5.2. *Given it is $HC_{\{iff\}}$, we have:*

1. *for each x_i, $k \in RS(x_i) \Leftrightarrow y_k \mapsto i$*
2. *for each x_i, $k \in PS(x_i) \Leftrightarrow i \in D_{y_k}$*
3. *$\nexists i \neq j, k$, such that $k \in RS(x_i)$ and $k \in RS(x_j)$*
4. *$\nexists i \neq j, k$, such that $k \in RS(x_i)$ and $k \in PS(x_j)$*

Proof. Points 1 and 2 follow from the definition of SI.

To prove point 3, suppose $\exists i, j, k$, such that $k \in RS(x_i)$ and $k \in RS(x_j)$, where $i \neq j$. By point 1, $y_k \mapsto i$ and $y_k \mapsto j$ simultaneously, which is a contradiction.

To prove point 4, suppose $\exists i, j, k$, such that $k \in RS(x_i)$ and $k \in PS(x_j)$, where $i \neq j$. By point 1 and point 2, $y_k \mapsto i$ and $j \in D_{y_k}$ is a contradiction. □

Points 3 and 4 explain that there is no sharing of values between (a) each pair of required sets and (b) each pair of required set and possible set of different variables. The following steps for constructing a complete assignment for $S_{X,Y}$ is used in subsequent proofs.

Construction 5.1. *Steps:*

1. $\forall x \in X$, *let* $RS'(x) = RS(x)$.

2. *a set* $R = \{1, \ldots, m\} - \bigcup_{x \in X} RS'(x)$

3. $\forall r \in R$, *pick a value* $d_r \in D(y_r)$, *and make* $RS'(x_{d_r}) = RS'(x_{d_r}) \cup \{r\}$

4. *we obtain the complete assignment* $e = \{x_j \mapsto RS'(x_j) \mid x_j \in X\} \cup \{y_k \mapsto j \mid x_j \in X, k \in RS'(x_j)\}$

Step 2 collects in the set R all indices of Y that are not in the required set of any variable in X. In other words, the variables in Y with indices in R are not assigned any value yet. Then step 3 picks an arbitrary value d_r from the domain of y_r for each $r \in R$ and fix y_r to d_r (by putting r into $RS'(x_{d_r})$). Note that by Property 5.2.2, r must be in $PS(x_{d_r})$ and thus $\forall x_j \in X, RS'(x_j) \subseteq PS(x_j)$ after step 3. Step 4 obtains a complete assignment e for $S_{X,Y}$ as a result.

Example 5.1. *Suppose* $X = \{x_1, x_2\}$, $Y = \{y_1, y_2, y_3\}$, $PS(x_1) = \{1, 2, 3\}$, $PS(x_2) = \{1, 3\}$, $RS(x_1) = \{2\}$, $RS(x_2) = \{\}$, $D_{x_1} = D_{x_3} = \{1, 2\}$, $D_{x_2} = \{1\}$. *Construction 5.1 may give us* $e = \{x_1 \mapsto \{2, 3\}, x_2 \mapsto \{1\}, y_1 \mapsto 2, y_2 \mapsto 1, y_3 \mapsto 1\}$ *as following steps.*

1. we make $RS'(x_1)$ and $RS'(x_2)$.

2. $R = \{1, 2, 3\} - \{2\} = \{1, 3\}$

3. we pick $2 \in D_{y_1}$ and $1 \in D_{y_3}$, and make $RS'(x_2) = \{\} \cup \{1\} = \{1\}$ and $RS'(x_1) = \{2\} \cup \{3\} = \{2, 3\}$

4. we obtain a complete assignment $e = \{x_1 \mapsto \{2,3\},\ x_2 \mapsto \{1\}\} \cup \{y_1 \mapsto 2,\ y_2 \mapsto 1,\ y_3 \mapsto 1\}$.

We first prove that *si* w.r.t. HC is as tight as *iff* or *ele* w.r.t. HC.

Theorem 5.11. $HC_{\{si\}} = HC_{\{iff\}}$.

Proof. $HC_{\{si\}} \geqslant HC_{\{iff\}}$ is trivially implied. To show the reverse by contradiction, suppose it is $HC_{\{iff\}}$ but not $HC_{\{si\}}$. Consider the following two cases: (1) by Lemma 5.1, $\exists i$ such that either (a) $x_i \mapsto PS(x_i)$ or (b) $x_i \mapsto RS(x_i)$ is not in any solution of *si*, (2) $\exists$ a value in the domain of y_i, say d_i, cannot be extended to any solution of *si*. (1)(a) Now we construct a complete assignment e by Construction 5.1 with doing $RS'(x_i) = PS(x_i)$ between step 1 and 2. Note that by Property 5.2.4, $\nexists j, k$ such that both $y_k \mapsto j$ and $y_k \mapsto i$ in e, where $j \neq i$. Here, e is a solution of *si*, which is a contradiction. (1)(b) Now we construct a complete assignment e by Construction 5.1 with an extra condition that each $d_r \neq i$ at step 3. Note that d_r must exist.[3] Again, e is a solution of *si*, which is a contradiction. (2) Note that $\{y_i \mapsto d_i, i \in x_{d_i}\}$ is $HC_{\{iff\}}$. We construct a complete assignment e by Construction 5.1 with doing $RS'(x_{d_i}) = RS'(x_{d_i}) \cup \{i\}$ between step 1 and 2. Note that by Property 5.2.2, $i \in PS(x_{d_i})$. Moreover by Property 5.2.4, $\nexists t \neq d_i$, such that $i \in RS(x_t)$. Thus, we have $y_i \mapsto d_i$ only. Again, e is a solution of *si*, which is a contradiction. From both of cases (1) and (2), this is a contradiction. □

Theorem 5.12. $HC_{\{ele\}} = HC_{\{iff\}}$.

Proof. First, we show $HC_{\{ele\}} \geqslant HC_{\{iff\}}$. Suppose it is $HC_{\{ele\}}$ but not $HC_{\{iff\}}$. Consider the following two cases: (1) by Lemma 5.1, $\exists\, i$ such that $x_i \mapsto PS(x_i)$

[3] Suppose d_r does not exist, thus $d_r = i$ and $D(y_r)$ must be equal to $\{i\}$, which essentially assign i to y_r. Then r must be in $RS(x_i)$ because of $HC_{\{iff\}}$, which is a contradiction with Construction 5.1.1.

or $RS(x_i)$ can not be extended to any solution of a constraint c, $j \in x_i \Leftrightarrow y_j = i$. (2) $\exists$ a value a in the domain of y_i which makes a constraint c, $y_i = j \Leftrightarrow i \in x_j$ to be not HC. (1) We consider the following cases: (a) $i \in D_{y_j}$ and $j \in RS(x_i)$ (b) $i \in D_{y_j}$ but $j \notin RS(x_i)$ (c) $i \notin D_{y_j}$ but $j \in RS(x_i)$ (d) $i \notin D_{y_j}$ and $j \notin RS(x_i)$ (a) Since $j \in x_{y_j}$ is HC, $j \in PS(x_i)$. Thus, $\{ x_i \mapsto PS(x_i), y_j \mapsto i \}$ and $\{ x_i \mapsto RS(x_i), y_j \mapsto i \}$ are two solutions of c. (b) Since $j \in x_{y_j}$ is HC, $j \in PS(x_i)$. Thus, $\{ x_i \mapsto PS(x_i), y_j \mapsto i \}$ is a solution of c. Let $b \in D_{y_j}$, we want to show $\exists b \neq i$. Suppose b must be i, by $j \in x_{y_j}$ is HC, $j \in RS(x_i)$, which is a contradiction. Thus $\{ x_i \mapsto RS(x_i), y_j \mapsto b \}$ is a solution of c. (c) This case is not possible, since $y_{x_i} = i$ is HC, $j \notin PS(x_i)$, which means $j \notin RS(x_i)$. (d) Since $y_{x_i} = i$ is HC, $j \notin PS(x_i)$, thus let $b \in D_{y_j}$, thus $\{ x_i \mapsto PS(x_i), y_j \mapsto b \}$ and $\{ x_i \mapsto RS(x_i), y_j \mapsto b \}$ are two solutions of c.

(2) We consider the folloing cases: (a) $a \neq j$ (b) $a = j$. (a) We want like to show that $i \notin RS(x_j)$. Suppose $i \in RS(x_j)$, since $y_{x_j} = j$ is HC, then y_i must equal to j, which contradicts to $a \neq j$. Thus $\{ x_j \mapsto RS(x_j), y_i \mapsto a \}$ is a solution of c. (b) Since $i \in x_{y_i}$ is HC, then $i \in PS(x_j)$. Then $\{ x_j \mapsto PS(x_j), y_i \mapsto a \}$ is a solution of c.

Combine the above two cases, this is an contradiction.

Second, we show $HC_{\{si\}} = HC_{\{ele\}} = HC_{\{iff\}}$. Since $HC_{\{si\}} \geqslant HC_{\{ele\}}$ is trivial, and we have $HC_{\{ele\}} \geqslant HC_{\{iff\}}$ already. By Theorem 5.11, we have $HC_{\{si\}}= HC_{\{iff\}}$, thus we have $HC_{\{si\}} = HC_{\{ele\}} = HC_{\{iff\}}$. □

Corollary 5.13. *$HC_{\{si\}}= HC_{\{c\}}$, where c can be iff or ele.*

Proof. Straightly followed by Theorem 5.11 and 5.12. □

Corollary 5.13 shows that the global implementation *si* gives no more pruning than *iff* or *ele* w.r.t. HC.

Theorem 5.14. $HC_{\{si\}}=HC_{\{\prod,si\}}$.

Proof. By Property 5.1.1, $HC_{\{\prod,si\}} \geqslant HC_{\{si\}}$. To show the reverse by contradiction, suppose it is $HC_{\{si\}}$ but not $HC_{\{\prod,si\}}$ due to a global partition constraint. Then, by Lemma 5.1, $\exists i$ such that either $x_i \mapsto PS(x_i)$ or $x_i \mapsto RS(x_i)$ cannot be extended to any solution of $\prod$ on X, but $\exists$ a solution $e = e_X \cup e_Y$ of si, where $e_X = \{x_1 \mapsto s_1, \ldots, x_i \mapsto s_i, \ldots, x_n \mapsto s_n\}$, $e_Y = \{y_1 \mapsto d_1, \ldots, y_m \mapsto d_m\}$, and $s_i = PS(x_i)$ or $RS(x_i)$. Note that e_X cannot be a solution of $\prod$ on X. Hence there are two cases, (1) $s_U = \bigcup_{i=1}^{n} s_i$, but $s_U \subset \{1, \ldots, m\}$. Then $\exists k$ such that $k \in \{1, \ldots, m\}$ but $k \notin s_U$. That means y_k does not take any value, this is a contradiction. (2) $\exists k_1, k_2$ such that $s_k = x_{k_1} \cap x_{k_2}$ and $s_k \neq \{\}$. Then $\exists k_3 \in s_k$. That means y_{k_3} need to take k_1 and k_2, this is a contradiction. From both of cases (1) and (2), this is a contradiction. □

Corollary 5.15. *$HC_{\{c\}} = HC_{\{a,c\}}$, where c can be si, iff, ele_X or ele_Y; and a can be $\prod$ or $\|$.*

Proof. In general, $HC_{\{\prod,si\}} \geqslant HC_{\{\|,si\}} \geqslant HC_{\{si\}}$. By Theorem 5.14, we have $HC_{\{\prod,si\}} = HC_{\{\|,si\}} = HC_{\{si\}}$. And we can easily derive the rest by Corollary 5.13 and Property 5.1.2, 5.1.3. □

Corollary 5.15 shows that any implementation of the SI channeling constraint subsumes all possible implementations of the partition constraints. In other words, the partition constraints can be removed from the model without losing constraint propagation strength.

5.3 Set-Set Channeling Constraints (SS)

Both M_X and M_Y are set models in this case. Channeling two set models imposes no characteristic constraints. The next property, which follows directly from the definition of SS, helps with our subsequent proofs.

Property 5.3. *Given it is $SBC_{\{iff\}}$, we have*

1. $j \in PS(x_i) \Leftrightarrow i \in PS(y_j)$
2. $j \in RS(x_i) \Leftrightarrow i \in RS(y_j)$

We are now ready to give a tightness relation between *ss* and *iff* w.r.t. SBC.

Theorem 5.16. $SBC_{\{ss\}}$=$SBC_{\{iff\}}$.

Proof. $SBC_{\{ss\}} \geqslant SBC_{\{iff\}}$ is trivially implied. To show the reverse by contradiction, suppose it is $SBC_{\{iff\}}$ but not $SBC_{\{ss\}}$. W.L.O.G., let it not be $SBC_{\{ss\}}$ on X (a symmetric proof can be made for Y). Then, by Lemma 5.1, $\exists i$ such that either (1) $x_i \mapsto PS(x_i)$ or (2) $x_i \mapsto RS(x_i)$ cannot be extended to any solution *ss*. For (1), We construct a complete assignment $e = \{x_i \mapsto PS(x_i) \mid x_i \in X\} \cup \{y_i \mapsto PS(y_i) \mid y_i \in Y\}$. Note that by Property 5.3.1, e is a solution of *ss*, which is a contradiction. For (2), We construct a complete assignment $e = \{x_i \mapsto RS(x_i) \mid x_i \in X\} \cup \{y_i \mapsto RS(y_i) \mid y_i \in Y\}$. Note that by Property 5.3.2, e is a solution of *ss*, which is a contradiction. From both of cases (1) and (2), this is a contradiction. □

Theorem 5.17. $SBC_{\{ele\}}$=$SBC_{\{iff\}}$.

Proof. First, we show $SBC_{\{ele\}} \geqslant SBC_{\{iff\}}$. Suppose it is $SBC_{\{ele\}}$ but not $SBC_{\{iff\}}$. Consider the following two cases: (1) by Lemma 5.1, $\exists$ i such that $x_i \mapsto PS(x_i)$ or $RS(x_i)$ can not be extended to any solution of a constraint c, $j \in x_i \Leftrightarrow i \in y_j$. (2) by Lemma 5.1, $\exists$ i such that $y_j \mapsto PS(y_j)$ or $RS(y_j)$ can not be extended to any solution of a constraint c, $j \in x_i \Leftrightarrow i \in y_j$. (1) We would like to show that (a) { $x_i \mapsto PS(x_i)$, $y_j \mapsto PS(y_j)$ } and (b) { $x_i \mapsto RS(x_i)$, $y_j \mapsto RS(y_j)$ } are two solutions of c. (a) If $j \in PS(x_i)$, since $i \in y_{x_i}$ is SBC, $i \in PS(y_j)$. If $j \notin PS(x_i)$, since $j \in x_{y_j}$ is SBC, $i \notin PS(y_j)$. Thus {

$x_i \mapsto PS(x_i)$, $y_j \mapsto PS(y_j)$ } is a solution of c. (b) if $j \in RS(x_i)$, since $i \in y_{x_i}$ is SBC, $i \in RS(y_j)$. If $j \notin RS(x_i)$, since $j \in x_{y_j}$ is SBC, $i \notin RS(y_j)$. Thus { $x_i \mapsto RS(x_i)$, $y_j \mapsto RS(y_j)$ } is a solution of c. (2) Symmetric proof can be made as (1).

This is an contradiction.

Second, we show $SBC_{\{ss\}} = SBC_{\{ele\}} = SBC_{\{iff\}}$. Since $SBC_{\{si\}} \geqslant SBC_{\{ele\}}$ is trivial, and we have $SBC_{\{ele\}} \geqslant SBC_{\{iff\}}$ already. By Theorem 5.16, we have $SBC_{\{si\}}= SBC_{\{iff\}}$, thus we have $SBC_{\{si\}} = SBC_{\{ele\}} = SBC_{\{iff\}}$. □

Corollary 5.18. *$SBC_{\{ss\}}$=$SBC_{\{c\}}$, where c can be iff or ele.*

Proof. Straightly followed by Theorem 5.16 and 5.17. □

Corollary 5.18 shows that the global implementation *ss* gives no more pruning than *iff* or *ele* w.r.t. SBC.

5.4 Int-Bool Channeling (IB)

We assume that M_X is an integer model with only one variable and M_Y is a Boolean model. Since the variable in X must be assigned exactly one value, channeling M_X and M_Y imposes the characteristic constraint on Y: $\sum_{y_i \in Y} y_i = 1$. We call this constraint *sum-to-one* and denote it by $\bigodot$.

The following property is for helping the following proofs.

Property 5.4. *Given $S_{x,Y}$ is $AC_{\{iff\}}$, we have:*

1. $x \mapsto i \Leftrightarrow y_i \mapsto 1$
2. $i \in D_x \Leftrightarrow 1 \in D_{y_i}$
3. $\nexists i \neq j$, *such that* $y_i \mapsto 1$ *(or* $D_{y_i} = \{1\}$*) and* $1 \in D_{y_j}$

4. if $\exists y_i \in Y$ *such that* $1 \in D_{y_i}$*, then* $\forall y_j \neq y_i \in Y$, $0 \in D_{y_j}$

Proof. To prove point 3. Suppose $\exists i \neq j$, such that $y_i \mapsto 1$ and $1 \in D_{y_j}$. By point 1 and point 2, $x \mapsto i$ and $j \in D_x$ is a contradiction.

To prove point 4. Suppose $\exists y_j$ such that $0 \notin D_{y_j}$, which means $D_{y_j} = \{1\}$. Then by point 3, this is a contradiction. □

Point 1 and 2 are from the definition of IB. Point 3 explains there can be only one variable in Y is assigned to 1. And Point 4 is a situation that derived from point 3.

Here, we prove that *ib* w.r.t. GAC is as tight as *iff* w.r.t. AC. The result follows directly from the fact that *ib* is actually the same as ele_Y, which in turn is a special case of the ele_Y in II (Boolean is a special case of integer).

Theorem 5.19. $GAC_{\{ib\}}$=$GAC_{\{ele\}}$=$AC_{\{iff\}}$.

Proof. We first prove for $GAC_{\{ib\}}$=$AC_{\{iff\}}$. $GAC_{\{ib\}} \geqslant AC_{\{iff\}}$ is trivially implied. To show the reverse by contradiction, suppose it is $AC_{\{iff\}}$ but not $GAC_{\{ib\}}$. Consider the following two cases: (1) $\exists$ a value $i \in D_x$, which is not $GAC_{\{ib\}}$. (2) $\exists$ a domain of y_i, say d_i, which is not $GAC_{\{ib\}}$. (1) Note that $i \in D_x$ and $1 \in D_{y_i}$ are $AC_{\{iff\}}$. Now we construct a complete assignment e in the following steps. First we make e contains $x \mapsto i$ and $y_i \mapsto 1$. Then by Property 5.4.4, for the rest of $y_j \in Y$, 0 must in D_{y_j}, and we make e contains $y_j \mapsto 0$. Hence e is a solution of *ib*, which is a contradiction. (2) Consider the following two cases (a) $d_i = 0$ and (b) $d_i = 1$. (a) $y_i \mapsto 0$ and $i \notin D_x$ are $AC_{\{iff\}}$. Now we construct a complete assignment e in the following steps. First we pick a value j such that $1 \in D_{y_j}$ and $j \neq i$, and make e contains $x \mapsto j$ and $y_j \mapsto 1$. Note that by Property 5.4.2 and $D_x \neq \{\}$, j must exist. Then by Property 5.4.4, for the rest of y_k, 0 must in D_{y_k}, and we make e contains $y_k \mapsto 0$. Again e is a solution fo *ib*, which is a contradiction. (b) $y_i \mapsto 1$ and

$x \mapsto i$ are $AC_{\{iff\}}$. Here, we have a same proof as (1). From both of cases (1) and (2), this is a contradiction.

Second, we prove for $GAC_{\{ib\}} = GAC_{\{ele\}} = AC_{\{iff\}}$. By Theorem 5.5, we have $GAC_{\{ele\}} \geqslant AC_{\{iff\}}$, since $Y \Rightarrow x$ can be consider as:

$$x_{y_i} = i, \forall y_i \in Y$$

Moreover, $GAC_{\{ib\}} \geqslant GAC_{\{ele\}}$ is trivial. Thus, together with $GAC_{\{ib\}} = AC_{\{iff\}}$, we have $GAC_{\{ib\}} = GAC_{\{ele\}} = AC_{\{iff\}}$. □

Theorem 5.20. **[CLS06]** $AC_{\{iff\}} = GAC_{\{iff,\odot\}}$.

Corollary 5.21. $GAC_{\{ib\}} = GAC_{\{ib,\odot\}}$.

Proof. By Theorem 5.19, $GAC_{\{ib\}} = AC_{\{iff\}}$. By Property 5.1.2, we can have $GAC_{\{ib,\odot\}} = AC_{\{iff,\odot\}}$. Thus, by Theorem 5.20 and Property 5.1.3, we have $GAC_{\{ib\}} = GAC_{\{ib,\odot\}}$. □

Theorem 5.20 and Corollary 5.21 show that the sum-to-one constraint does not cause any more domain reduction when working with either *si* or *iff*.

5.5 Set-Bool Channeling (SB)

We assume that M_X is a set model with only one variable and M_Y is a Boolean model.

The following property is for helping the following proof.

Property 5.5. *Given* $S_{x,Y}$ *is* $SBC_{\{iff\}}$, *we have:*

1. $i \in PS(x) \Leftrightarrow 1 \in D_{y_i}$
2. $i \notin PS(x) \Leftrightarrow y_i \mapsto 0$ $(D_{y_i} = \{0\})$
3. $i \in RS(x) \Leftrightarrow y_i \mapsto 1$ $(D_{y_i} = \{1\})$

4. $i \notin RS(x) \Leftrightarrow 0 \in D_{y_i}$

Point 1 and 3 are from the definition of SB. Point 2 is equivalent to point 1, and point 4 is equivalent to point 3.

Here, we prove that *sb* w.r.t. HC is as tight as *ele* w.r.t. HC, and as tight as *iff* w.r.t. HC.

Theorem 5.22. $HC_{\{sb\}}=HC_{\{ele\}}=HC_{\{iff\}}$.

Proof. We first prove $HC_{\{sb\}}=HC_{\{iff\}}$. $HC_{\{sb\}}\geqslant HC_{\{iff\}}$ is trivially implied. To show the reverse by contradiction, suppose it is $HC_{\{iff\}}$ but not $HC_{\{sb\}}$. By Lemma 5.1, consider the following two cases: (1)(a) $x \mapsto PS(x)$ or (b) $x \mapsto RS(x)$ is not $HC_{\{sb\}}$. (2) $\exists$ a domain of y_i, say d_i, which is not $GAC_{\{sb\}}$. (1)(a) Note that for each $k \in PS(x), y_k \mapsto 1$ is $HC_{\{iff\}}$. Now we construct a complete assignment e in the following steps. First we make $e = \{x \mapsto PS(x)\} \cup \{y_k \mapsto 1 \mid k \in PS(x)\}$. Then for the rest of y_l which is not assigned with value yet, make e contains $y_l \mapsto 0$. Note that by Property 5.5.2, $0 \in D_{y_l}$). Hence e is a solution of *sb*, this is a contradiction. (1)(b) Note that for each $k \in RS(x), y_k \mapsto 1$ is $HC_{\{iff\}}$. Now we construct a complete assignment e in the following steps. First we make $e = \{x \mapsto RS(x)\} \cup \{y_k \mapsto 1 \mid k \in RS(x)\}$. Then for the rest of y_l which is not assigned with value yet, make e contains $y_l \mapsto 0$. Note that by Property 5.5.4, $0 \in D_{y_l}$. Again e is a solution of *sb*, this is a contradiction. From both of cases (a) and (b), this is a contradiction. (2) Consider the following two cases: (a) $d_i = 0$ and (b)$d_i = 1$. (a) $y_i \mapsto 0$ and $i \notin PS(x)$ (and $i \notin RS(x)$) are $HC_{\{iff\}}$. Now we construct a complete assignment e same as (1) (a). And e is a solution of *sb*, this is a contradiction. (b) $y_i \mapsto 1$ and $i \in PS(x)$ are $HC_{\{iff\}}$. Here, we have a same proof as (1)(a). From both of cases (1) and (2), This is a contradiction.

Second, we prove for $HC_{\{sb\}} = HC_{\{ele\}} = HC_{\{iff\}}$. By Theorem 5.12, we have $HC_{\{ele\}} \geqslant HC_{\{iff\}}$, since $Y \Rightarrow x$ can be consider as:

$$i \in x_{y_i}, \forall y_i \in Y$$

Moreover, $HC_{\{sb\}} \geqslant HC_{\{ele\}}$ is trivial. Thus, together with $HC_{\{sb\}} = HC_{\{iff\}}$, we have $HC_{\{sb\}} = HC_{\{ele\}} = HC_{\{iff\}}$. □

5.6 Discussion

In ideal situation, if a solver provides *gio* (i.e. *ii, si, ss, sb*, and *ib*) or *ele*, they should be maintained HC. While in real situation, it is not always true. For example, ILOG solver provides `IlcInverse` as *ii*, but `IlcInverse` is just maintained a equivalent consistency level as maintaining *AC* on each constraint in *iff*. Another example is using element constraint for int-int channeling. From the user manual of SICStus Prolog:

> element(?X, +*List*, ?Y)
>
> ...
>
> element/3 maintains **domain-consistency** in X and **interval-consistency** in List and Y.
>
> ...
>
> A domain constraint is an expression $X :: I$, where X is a domain variable and I is a nonempty set of integers. A set S of domain constraints is called a store. $D(X, S)$, the domain of X in S, is defined as the intersection of all I such that $X :: I$ belongs to S.
>
> ...
>
> A constraint C is *domain-consistent* wrt. S iff, for each variable X_i and value V_i in $D(X_i, S)$, there exist values V_j in $D(X_j, S), 1 \leq j \leq, i \neq j$, such that $C(V_1, ..., V_n)$ is true.

> ...
>
> A constraint C is *interval-consistent* wrt. S iff, for each variable X_i and value V_i in $D(X_i, S)$, there exist values V_j and W_j in $D'(X_j, S)$, $1 \leq j \leq n, i \neq j$, such that $C(V_1, \ldots, min(D(X_i, S)), \ldots, V_n)$ and $C(W_1, \ldots, max(D(X_i, S)), \ldots, W_n)$ are both true.

Although the other solvers that we investigated in the previous chapter do not state the consistency level they maintain, element constraint is usually not maintained as GAC because of the performance issue.

Our theoretic result shows that except for II, maintaining a higher level of consistency on the entire global channeling constraint does not increase the pruning power, which is an useful information on implementing efficient channeling constraints.

Chapter 6

Algorithms and Implementation

In the previous chapter, we investigate and report the comparison on consistency levels among the various realizations for each of the channeling constraint. A major result is that, except for II, a global constraint maintaining HC on the entire channeling constraint gives the same pruning power as maintaining HC on each of the constraints in an *iff* implementation. One might be tempted to conclude that (a) the *iff* implementations are the best for the SI, SS, IB, and SB channeling constraints, and (b) a GAC global constraint implementation is the best for the II constraint. For (a), we are going to show that the *iff* implementations are inefficient since there are a large number of constraints. During constraint propagation, many invocations of propagators are proved to be unnecessary. For (b), we have so far been unable to devise an efficient propagator for the global II constraint to enforce GAC. Apparently, the cost for maintaining GAC is so high that it cannot be compensated by the extra pruning achieved. The implementation details is reported in the last section in this chapter, and experimental results are given in the next chapter.

The discussion above should not be used as arguments against global constraint implementations, since we can always maintain a lower level of consistency than HC for a global constraint. An important advantage of global constraint implementation is that information from many constraints can be considered in one go, providing a more complete view and saving time for coordinating the domain reduction and propagation of pruning information among a large number of constraints. In the following, we analyze the inefficiency of the *iff* implementations, followed by presentations of two generic propagators for making part of and the complete set of the *iff* constraints into global constraints.

6.1 Source of Inefficiency

If we are channeling models M_X and M_Y with n and m variables respectively, there should be nm *iff* constraints, each of which is associated with a propagator, and each propagator is invoked whenever there is domain reduction. Consider a situation in II, in which the value 3 is removed from D_{x_1}. If we are maintaining AC on the individual *iff* constraints, this information will invoke m of the *iff* propagators involving x_1, but only one propagator takes effect and removes the value 1 from D_{y_3}. This last domain reduction in turn trigger the other $n-1$ propagators involving y_3, but no reduction will happen. Suppose, in SI, that 1 is added to $RS(x_3)$. If we are maintaining HC on the individual *iff* constraints, this would invoke the m propagators involving x_3, and only one would take effect and cause 3 to be assigned to y_1. The assignment is equivalent to removing values $\{1, 2, 4, \ldots, n\}$ from D_{y_1}, which would in turn invoke $n-1$ propagators involving y_1 and cause $x_1 \not\leftrightarrow 1$, $x_2 \not\leftrightarrow 1$, $x_4 \not\leftrightarrow 1$, ..., $x_n \not\leftrightarrow 1$. Since $n-1$ X variables are updated, $(n-1)(m-1)$ propagators involving these variables will be invoked without further reduction effect. From these two examples, we can see that usually a large number of invocations of propagators is unnecessary and wasteful of computing resources. Similar analysis leads to the *iff* column in Table 6.1, which reports the big O order of

Type	Task	*iff*	*ele*	*glo*
II	VD	$O(nm)$	$O(n+m)$	$O(n+m)$
	DR	$O(n+m)$	$O(n+m)$	$O(1)$
SI	VD	$O(nm)$	$O(n+m)$	$O(n)$
	DR	$O(n+m)$	$O(n+m)$	$O(1)$
SS	VD	$O(n+m)$	$O(n+m)$	$O(1)$
	DR	$O(n+m)$	$O(n+m)$	$O(1)$
IB	VD	$O(m)$	$O(m)$	$O(m)$
	DR	$O(m)$	$O(1)$	$O(1)$
SB	VD	$O(m)$	$O(1)$	$O(1)$
	DR	$O(m)$	$O(1)$	$O(1)$

Table 6.1: Big O Order of Propagator Invocations

the number of propagator invocations for various implementations and channeling constraint types. The table gives the number of propagator invocations caused by both variable decisions (VD) and domain reductions (DR) for each channeling constraint.

6.2 Generalized Element Constraint Propagators

Cheng et al. [CCLW99] suggest using the `element` constraint as a more succinct and compact way of expressing the II channeling constraint. This would work also for IB, but not for SI, SS, and SB which involve set variables. We propose a generalized `element` constraint for both integer and set variables specialized for implementing channeling constraints. The form of the constraint is `gElement(`I`,[`$V_1,\ldots,V_n$`],`c`)`, where I and V_i's are either integer (Boolean) or set variables and c is an integer constant. The new constraint

xDomRed(i : index of variable x) ▷ be invoked when the domain of x_i is changed
    if v is impossible for x_i **then**
        $y \not\leadsto i$

yDomRed(rm: set of new impossible values for y;
        rq : set of new decided values for y;) ▷ be invoked when the domain of y is changed
    for each $j \in rq$ **do**
        $x_j \leadsto v$

Figure 6.1: The Propagator for `gElement` of the form $x_y = v$ or $v \in x_y$

is a generalization of `element` since set variables are now supported. It is a specialization (for efficient implementation) since c must be a constant.

When I and V_i's are integer variables, `gElement` has the same meaning as `element`. When I is a set variable and V_i's are integer variables, the constraint enforces that $\forall j \in I$, $V_j = c$. When I is an integer variable and V_i's are set variables, the constraint means that $c \in V_I$. When both I and V_i's are set variables, the semantics is that $\forall j \in I$, $c \in V_j$. When V_i's are integer variables, the constraint is abbreviated to $V_I = c$. When V_i's are set variables, the constraint is abbreviated to $c \in V_I$. Suppose the variable x_3 is instantiated to the set $\{2, 4, 7\}$ in SI. Both the *ele* constraints would enforce y_2, y_4, and y_7 to take the value 3, and *vice versa*.

Figure 6.1 gives the pseudocode of the propagator for the `gElement` constraint of the form either $x_y = v$ or $v \in x_y$ (i.e. x_i's are the principal variables). By making use of notions and notations defined in Chapter 2, the pseudocode is generic in the sense that the different combinations of variable types are immaterial in understanding the algorithms. The propagator consists of two procedures: xDomRed is invoked when one of the x_i variables is updated and yDomRed is invoked when the y variable is updated. The procedure xDomRed is called with the index i of the updated variable x_i. Depending on the status of the value v with respect to D_x, D_y is updated accordingly. On the other hand, yDomRed is called with the new impossible values and/or the new decided value for y as a result of the last update. Based on these values, domains of the appropriate x_j variables are updated.

Note that Boolean mapping constraint Y_x is actually a special case of $x_{y_i} = i$, in which our `gElement` Propagator is also fit for it.

Property 6.1. *A reified constraint $C_1 \Leftrightarrow C_2$ is satisfied if and only if both C_1 and C_2 are true or both C_1 and C_2 are false, where C_1 and C_2 are constraint. Thus, we have propagation rules of (1) C_1 is true $\Rightarrow C_2$ is true, (2) C_2 is true $\Rightarrow C_2$ is true, (3) C_1 is false $\Rightarrow C_2$ is false, (4) C_2 is false $\Rightarrow C_2$ is false.*

Proof. It is by definition of reified constraint. □

Lemma 6.1. *The iff constraint can be maintained as HC by $4mn$ propagation rules (by Property 6.1), they are: (1) $x_i \leadsto j \Rightarrow y_j \leadsto i$; (2) $y_j \leadsto i \Rightarrow x_i \leadsto j$; (3) $x_i \not\leadsto j \Rightarrow y_j \not\leadsto i$; (4) $y_j \not\leadsto i \Rightarrow x_i \not\leadsto j, \forall x_i \in X, \forall y_j \in Y$.*

Proof. It is straightly followed by the definition of HC. □

Theorem 6.2. *Using the* `gElement` *propagator in each constraint in ele is equivalent as maintains HC on each constraint in iff.*

Proof. By 8Lemma 6.1, there are propagation rules (1), (2), (3) and (4). From Figure 6.1, all the rules in (1) and (4) are handled by procedure "xDomRed", and all the rules in (2) and (3) are handled by procedure "yDomRed". □

Similar analysis is performed to give the big O order of the number of `gElement` propagator invocations for the *ele* implementation in Table 6.1. Actually, the number of propagator invocations is proportional to the number of constraints with their variables' domains are changed.

6.3 Global Channeling Constraint

In Chapter 4, we introduce the existing global channeling constraints in different solvers. While they are for int-int channeling constraint (II), but not for SI, SS, SB and IB which involve set variables and Boolean variables. On the other hand, some solvers do not provide an implementation of II which is maintained GAC. In this section, we present two algorithms on global channeling constraint. One is the generalization of those existing global channeling constraints for integer, set and Boolean variables. This generalization maintains a consistency level as same as maintaining HC on each constraint in *iff*. Another one is an implementation of II which is maintained GAC, and it is based on the implementation of global `AllDiff` constraint.

6.3.1 Generalization of Existing Global Channeling Constraints

From Table 6.1, we can see that the ele_X and ele_Y implementations offer a good reduction in number of propagator invocations. This good trend suggests to go one step further to bundle all *iff* constraints (and thus also all *ele* constraints) into one global constraint as our *glo* implementation. Figure 6.2 gives the pseudocode of the *glo* propagator for channeling models M_X and M_Y. Again, the pseudocode is generic in the sense that it is applicable to all five channeling constraints. The *glo* propagator has three procedures: domRed is a common procedure called by xDomRed and yDomRed, which are invoked by updates of an x_i or y_j variable. Arguments to the xDomRed and yDomRed procedures include the index of the updated variable, and the new impossible and/or decided values for the updated variable as a result of the last update. Upon entry, the xDomRed and yDomRed procedures simply pass the variables to

domRed(Z : set of variables $\{z_1, \ldots z_n\}$; ▷ be invoked by xDomRed or yDomRed
$\quad v$: value;
$\quad rm$: set of impossible values;
$\quad rq$: set of decided values)
$\quad$ **for each** $j \in rm$ **do**
$\quad\quad z_j \not\rightsquigarrow v$
$\quad$ **for each** $j \in rq$ **do**
$\quad\quad z_j \rightsquigarrow v$

---

xDomRed(i : index of variable x; ▷ be invoked when the domain of x_i is changed
$\quad rm$: set of new impossible values for x_i;
$\quad rq$: set of new decided values for x_i)
$\quad$ domRed(Y, i, rm, rq)

---

yDomRed(i : index of variable y; ▷ be invoked when the domain of y_i is changed
$\quad rm$: set of new impossible values for y_i;
$\quad rq$: set of new decided values for y_i)
$\quad$ domRed(X, i, rm, rq)

Figure 6.2: The *glo* Propagator

be updated and the received arguments to domRed. Based on the received values, the domRed procedure updates the appropriate variables accordingly.

Theorem 6.3. *The glo propagator maintains HC on the iff constraints.*

Proof. By Lemma 6.1, there are propagation rules (1), (2), (3) and (4). From Figure 6.2, all the rules in (1) and (3) are handled by procedure "xDomRed", and all the rules in (2) and (4) are handled by procedure "yDomRed". □

Table 6.1 gives also the big O order of the number of *glo* propagator invocations in the last column. We can see that the *glo* propagator in general gives a drastic improvement in performance over the *iff* and *ele* propagators. There are a few points to note. First, for IB and SB, ele_Y contains only one constraint, which is equivalent in pruning behavior to the *glo* constraint. That is why they share the same big O order. Second, IB and SB are special cases of II and SI respectively. The big O order entries of IB and SB can be obtained from those of II and SI by setting n to 1 (since $|X| = 1$ for both IB and SB).

6.3.2 Maintaining GAC on Int-Int Channeling Constraint

In this section, we give an algorithm for maintaining GAC on global II (gII), which is based on matching theories and Régin's all-difference algorithm [Rég94]. In the following, we are channeling two integer models M_X and M_Y with variables $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ respectively.

Method 6.1. *Construct a bipartite graph $G_{ii} = (V, E)$, where $V = X \cup Y$ ($x_i \in X$ on the left and $y_j \in Y$ on the right) and $E = \{\{x_i, y_j\} \mid j \in D_{x_i}$ and $i \in D_{y_j}\}$.*

Figure 6.3 shows a result G_{ii} that is constructed by Method 6.1 for $X = \{x_1, \ldots x_4\}$, $Y = \{y_1, \ldots y_4\}$, $D_{x_1} = D_{y_1} = D_{y_2} = \{1, 2\}$, $D_{x_2} = \{1, 2, 3\}$, $D_{x_3} =$

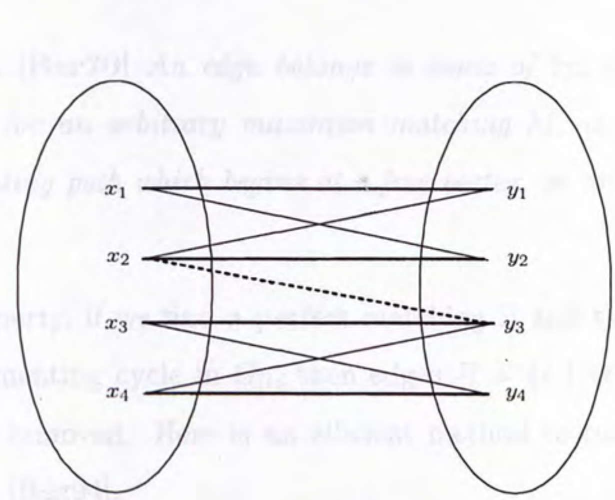


Figure 6.3: Perfect Matching

$D_{x_4} = D_{y_4} = \{3, 4\}$ and $D_{y_3} = \{2, 3, 4\}$. In this figure, the bold edges are a perfect matching Ξ of G_{ii}. By considering each edge $\{x_i, y_j\}$ as an assignment $\{x_i \mapsto j, y_j \mapsto i\}$, we can clearly obtain a solution s_Ξ of *ii*: $\{x_i \mapsto i \mid x_i \in X\} \cup \{y_i \mapsto i \mid y_i \in Y\}$. By this example, we have the following theorem and corollary:

Theorem 6.4. *Given G_{ii} is constructed by Method 6.1, ii has a solution s_Ξ if and only if G_{ii} has a perfect matching Ξ.*

Corollary 6.5. *Given G_{ii} is constructed by Method 6.1, $\exists$ a perfect matching of G_{ii} contains an edge $\{x_i, y_j\}$ if and only if $\{x_i \mapsto j, y_j \mapsto i\}$ can be extended to a solution of ii, where $j \in D_{x_i}$ and $i \in D_{y_j}$.*

Theorem 6.4 gives us a method of finding a solution for a given global II, and Corollary 6.5 points out a condition on when a domain value (in both models) can be extended to a solution. If we have an efficient way to remove all edges that are not in any perfect matching, then we can maintain GAC on global II. Here is a property helps us.

Property 6.2. [**Ber70**] *An edge belongs to some of but not all maximum matchings, iff, for an arbitrary maximum matching M, it belongs to either an even alternating path which begins at a free vertex, or an even alternating cycle.*

By this property, if we find a perfect matching Ξ and the set of edges Θ of all even augmenting cycle in G_{ii}, then edges $R = \{e \mid \forall e \in E, e \notin \Xi$ and $e \notin \Theta\}$ can be removed. Here is an efficient method to remove R, same as what Régin did [Rég94].

Method 6.2. *Given a bipartite graph $G = (V, E)$, where $V = X \cup Y$, and a perfect matching Ξ of G, construct an oriented graph $G' = (V, E')$, where $E' = \{(x, y) \mid \forall\{x, y\} \in \Xi, x \in X$ and $y \in Y\} \cup \{(y, x) \mid \forall\{x, y\} \in E - \Xi, x \in X$ and $y \in Y\}$. If the set of edges Θ' are edges of all strongly connected components of G', then edges $R' = \{\{x, y\} \mid \forall\{x, y\} \in E, \{x, y\} \notin \Xi, (x, y)$ and $(y, x) \notin \Theta'\}$ can be removed.*

Method 6.2 is efficient, as finding all strongly connected components takes $O(|V| + |E|)$ steps. Note that the direction of edges in E' makes any path traversal forming an alternating path. Thus if $\exists$ an edge e in a strong connected component, then there must be an even alternating cycle (a cycle in bipartite graph must be even) contains e, and *vice versa*. Hence, we have $R = R'$. The dotted edge $\{x_2, y_3\}$ in Figure 6.3 is an example that it does not belong to any perfect matching (solution) in the graph.

Let us summarize our method of maintaining GAC on global II (gII), and calculate the overall complexity.

1. Construct a bipartite graph G from X and Y, remove domains that can't form edges.

2. Find a perfect matching Ξ of G, no solution can be found if this fails.

3. Construct another graph G', by orienting edges $\{x_i, y_j\}$ belongs to Ξ as (x_i, y_j), or orienting as (y_j, x_i) otherwise.

4. Find edges Θ of all strongly connected components of G'.

5. Remove the domains of the corresponding edges $e \notin \Theta$.

Step 1, 3, 4, 5 takes $O(|V| + |E|)$ steps, step 2 takes $O((|V| + |E|)\sqrt{|V|})$ steps. Thus the overall complexity is $O((|V| + |E|)\sqrt{|V|})$.

In practice, step 1,2 and 3 can be built once, and they can be maintained during search. Step 1 and 3 can be maintained by propagators in Figure 6.2. While on maintaining step 2, if k edges are removed in the matching, then $O(\sqrt{k}|E|)$ steps are need for repair.

Chapter 7

Experiments

To evaluate the feasibility and efficiency of our proposed propagator algorithms, we have implemented the propagators and compared them against techniques utilizing available constraints in existing solvers. One way to perform benchmarking is to construct a combined model with only variables and channeling constraints. Random variable assignments and pruning can then be generated to exercise the various implementations and observe their performances. Such an approach is *ad hoc* in the least. We test our implementations on real CSP benchmarks from the CSPLib. Smith [Smi01] suggests the models $\{Q_c, Q_r, Q_z\}$ of the n-queens problem, $\{L_n, L_p, L_z\}$ of the Langford's problem, and $\{G_g, G_p, G_w, G_z\}$ of the Social Golfers problem. The models $\{A_n, A_p, A_z\}$ of the All Interval Series problem, $\{B_c, B_p, B_z\}$ of the Balanced Academic Curriculum, and $\{S_n, S_p, S_z\}$ of the Steiner Triple Systems are by Choi et al. [CLS06], Hnich et al. [HKW02], and Law and Lee [LL06] respectively.

For each problem, we test a wide range of instances which terminate in reasonable time. All executions search for all solutions to exercise the channeling constraints to the fullest, using smallest domain first and first unbound variable heuristics for integer variables and set variables respectively. All experiments are conducted using ILOG Solver 4.4 on a Sun Blade 25000 workstation with 2GB memory.

In the resulting tables, each row corresponds to a problem instance, and each column corresponds to a type of channeling constraint implementation. In the same table, if all the channeling constraint implementations maintain the same consistency level, we report their fails in the rightmost column. On the other hand, if the implementations maintain different consistency levels, we group them into blocks according to their consistency levels, and report their fails in the rightmost within each block. Each table caption specifies the models used for channeling. Variables of the bolded model is used as search variables. The aim of the experiments, except those relate to maintaining GAC on int-int channeling constraint, is to compare the runtime of the *glo* implementation against all other implementations. Thus, despite reporting the runtime on each type of channeling constraints implementation, speedups (runtime of an implementation / runtime of *glo* implementation), are reported at the bottom of each implementation, i.e. the bottom of table. Speedups are averaged over the number of instances, specified at the right bottom corner, which run more than one second on their *iff* implementation [1] . We report also in brackets the standard deviation of each statistics.

7.1 Int-Int Channeling Constraint

Theorem 5.6 in Chapter 5 tells us about maintaining GAC on a global int-int channeling constraint causes more domain reduction. Thus we divide this section into two subsections. The first focuses on the implementations that are equivalent to maintaining AC on each *iff* constraints. This compares the runtime among the *gElement* implementation, *glo* implementation, and those

[1] From our experience, the runtime report by iLog solver may not be accurate. There can be $+/-0.1 \sim 0.2$ variation in second. Thus we want to minimize the error for calculation in this way

predefined constraints in ILOG Solver. The second focuses on the implementations that are equivalent to maintaining GAC on the global int-int channeling constraint. This compares the runtime among the gII implementations and those predefined constraints in ILOG Solver.

7.1.1 Efficient AC implementations

Tables 7.1, 7.2, 7.3, 7.4, and 7.5 report the results for int-int channeling between models $\mathbf{Q_c}$ and Q_r, $\mathbf{L_n}$ and L_p, L_n and $\mathbf{L_p}$, $\mathbf{A_n}$ and A_p and A_n and $\mathbf{A_p}$ respectively. The result for channeling between models Q_c and $\mathbf{Q_r}$ are identical to the one of channeling between models $\mathbf{Q_c}$ and Q_r; thus we leave it out. The *iff* implementation is the basic one. Hnich et al. [HSW04] prove that keeping pairwise disequality ($\neq$) constraints on either model does not increase pruning. We study how the extra disequality constraints in the implementation $\neq$ *iff* $\neq$ can degrade performance. For the realization of pairwise disequality ($\neq$) constraints, we use the `IlcAllDiff` constraint, which is a predefined constraint in ILOG Solver. The `IlcAllDiff` constraint has an option for choosing different consistency levels, and we choose the one that is equivalent as maintaining AC on each pairwise disequality constraint. The *ele* implementations use `gElement` with variables in models 1 and 2 together. For the II case, the ILOG `element` constraint can also be used, we use ele_{12} implementation to represent ILOG implementation, where 1 is the letter representing variable in model 1, and 2 is the letter representing variable in model 2. ILOG Solver also provides the `IlcInverse` constraint, which is also a global constraint maintaining the same consistency (by our experimental observation only) as *glo*. Their performances are *basically identical* and we leave out the results.

Results in Table 7.1, 7.2, 7.3, 7.4, 7.5 confirm that *glo* are the fastest among all implementations. The speedups for the $\neq$ *iff* $\neq$ implementation

n	$\neq$ *iff* $\neq$	*iff*	ele_{cr}	*ele*	*glo*	Fails
8	0.03	0.03	0.03	0.02	0.01	256
9	0.15	0.13	0.11	0.08	0.06	929
10	0.57	0.58	0.43	0.33	0.23	4106
11	2.65	2.65	1.92	1.41	0.99	17601
12	13.41	13.37	9.27	6.89	4.74	80011
13	71.9	71.41	48.41	35.89	23.82	392128
14	412.67	409.19	265.26	198.2	128.2	2101047
15	2508.1	2494.85	1569.64	1178.39	741.85	11724826
16	16527.7	16366.6	9888.31	7482.33	4593.78	70692998
Speedup	3.12(0.35)	3.1(0.33)	2.04(0.09)	1.52(0.08)	1(0)	6

Table 7.1: Result for int-int channeling between models $\mathbf{Q_c}$ and Q_r of the N-Queens Problem

are ranging from 2.6 to 3.51. Moreover, the *ele* implementation are always faster than ele_{12} provided by ILOG Solver. The $\neq$ *iff* $\neq$ implementation is usually the slowest, but sometimes the *iff* implementation can be a little bit slower. The $\neq$ *iff* $\neq$ implementation should be slower than *iff* due to the extra work load by the pairwise disequality constraints. In real situation, if the implementation of $\neq$ is efficient, like the one we used (IlcAllDiff), it is possible to reduce the number of propagation steps that should be done by the "inefficient" *iff*. Thus, the instances $L(10,4), L(11,4), \ldots, L(15,4)$ in Table 7.3 have the $\neq$ *iff* $\neq$ implementation slightly faster than the *iff* implementation.

7.1.2 GAC Implementations

Tables 7.6, 7.7, 7.8, 7.9, and 7.10 report the results for int-int channeling between models $\mathbf{Q_c}$ and Q_r, $\mathbf{L_n}$ and L_p, L_n and $\mathbf{L_p}$, $\mathbf{A_n}$ and A_p and A_n and $\mathbf{A_p}$ respectively. Our implementation gII maintains GAC on ii. By Corollary 5.8, we form three other implementations: $\forall\, iglo\, \forall$, $\forall\, iglo$, and $iglo\, \forall$, which achieve

n, k	$\neq$ *iff* $\neq$	*iff*	ele_{np}	*ele*	*glo*	Fails
7,2	0.03	0.03	0.02	0.01	0.01	93
8,2	0.14	0.13	0.09	0.07	0.04	340
9,2	0.88	0.87	0.58	0.44	0.26	2800
10,2	4.9	4.78	3.05	2.42	1.39	13345
11,2	40.69	39.65	23.77	19.59	10.94	71984
12,2	274.5	265.33	155.74	130.27	71.53	438141
7,3	0.02	0.02	0.02	0.02	0.01	22
8,3	0.06	0.05	0.05	0.03	0.02	61
9,3	0.26	0.25	0.21	0.14	0.09	257
10,3	1.01	0.92	0.75	0.53	0.31	788
11,3	4.09	3.88	3.06	2.21	1.27	2977
12,3	21.5	20.25	15.88	11.58	6.6	13687
13,3	121.84	116.25	88.41	65.45	37.37	69376
14,3	557.15	530.43	397.27	297.88	169.59	281728
7,4	0.02	0.02	0.01	0.01	0	8
8,4	0.05	0.05	0.04	0.02	0.02	23
9,4	0.11	0.11	0.12	0.06	0.04	44
10,4	0.33	0.32	0.33	0.19	0.12	130
11,4	1.22	1.16	1.13	0.69	0.41	414
12,4	5.1	4.9	4.28	2.66	1.6	1344
13,4	23.97	22.85	17.18	11.19	6.69	5111
14,4	112.79	99.36	64.87	43.06	25.86	16944
15,4	455.57	438.92	245.39	163.73	96.14	59479
Speedup	3.61(0.49)	3.45(0.45)	2.38(0.17)	1.73(0.05)	1(0)	12

Table 7.2: Result for int-int channeling between models $\mathbf{L_n}$ and L_p of the Langford's Problem

n, k	$\neq$ *iff* $\neq$	*iff*	ele_{np}	*ele*	*glo*	Fails
7,2	0.02	0.03	0.02	0.01	0.01	82
8,2	0.13	0.12	0.08	0.06	0.04	291
9,2	0.78	0.77	0.46	0.4	0.24	2575
10,2	4.46	4.37	2.5	2.23	1.31	12531
11,2	36.84	36.11	19.33	18.52	10.49	67765
12,2	246.87	240.4	125.69	123.11	68.55	405667
7,3	0.03	0.03	0.02	0.02	0.01	31
8,3	0.05	0.06	0.04	0.03	0.02	54
9,3	0.27	0.27	0.2	0.14	0.08	205
10,3	0.88	0.85	0.6	0.48	0.28	646
11,3	3.7	3.59	2.48	2.03	1.18	2426
12,3	17.04	16.43	10.81	9.36	5.42	9923
13,3	89.78	87.76	55.61	49.31	28.75	47416
14,3	376.56	365.45	225.25	205.23	120.4	173295
7,4	0.03	0.03	0.03	0.01	0.01	10
8,4	0.08	0.07	0.06	0.04	0.03	25
9,4	0.17	0.17	0.14	0.1	0.06	56
10,4	0.48	0.49	0.39	0.28	0.18	138
11,4	1.12	1.14	0.89	0.66	0.41	272
12,4	4.85	4.92	3.45	2.67	1.64	947
13,4	18.54	18.59	11.19	8.72	5.42	2628
14,4	63.93	63.98	33.34	26.7	16.64	7302
15,4	239.61	242.35	111.33	88.84	54.56	22775
Speedup	3.41(0.4)	3.36(0.43)	1.97(0.1)	1.68(0.05)	1(0)	12

Table 7.3: Result for int-int channeling between models L_n and $\mathbf{L_p}$ of the Langford's Problem

n	$\neq$ *iff* $\neq$	*iff*	ele_{np}	*ele*	*glo*	Fails
8	0.02	0.02	0.02	0.01	0.01	104
9	0.08	0.07	0.07	0.04	0.04	349
10	0.34	0.33	0.29	0.2	0.15	1298
11	1.57	1.49	1.25	0.91	0.65	5136
12	7.78	7.39	6	4.39	3.11	22238
13	40.27	38.03	30.64	21.44	15.5	101463
14	221.5	208.18	165.12	116.34	83.33	495826
15	1280.86	1201.76	940.02	693.96	472.83	2558523
16	7798.82	7331.12	5683.88	4083.17	2850.82	14099360
Speedup	2.6(0.12)	2.46(0.11)	1.97(0.03)	1.44(0.03)	1(0)	6

Table 7.4: Result for int-int channeling between models $\mathbf{A_n}$ and A_p of the All Interval Series Problem

n	$\neq$ *iff* $\neq$	*iff*	ele_{np}	*ele*	*glo*	Fails
9	0.03	0.02	0.02	0.02	0.02	120
10	0.08	0.07	0.06	0.04	0.03	324
11	0.26	0.25	0.17	0.15	0.11	981
12	0.87	0.86	0.56	0.49	0.35	3146
13	3.23	3.2	2.01	1.8	1.27	10892
14	13.12	13.03	8.05	7.3	5.09	40352
15	60.14	59.6	36.54	33.85	23.18	173549
16	299.78	298.53	182.08	170.19	114.67	794100
17	1710.95	1700.09	1044.36	969.97	643.59	4162212
Speedup	2.58(0.06)	2.56(0.06)	1.59(0.02)	1.45(0.04)	1(0)	6

Table 7.5: Result for int-int channeling between models A_n and $\mathbf{A_p}$ of the All Interval Series Problem

the same domain reduction as gII, where *iglo* represents the `IlcInverse` constraint of ILOG Solver. For the realization of the global `allDiff` ($\forall$) constraints, we use the `IlcAllDiff` constraint, and we set its consistency level to maintain GAC. On the right-hand-side of each table, we append the results of *glo* to give a better overall picture on the int-int channeling constraints implementation. The bolded values in each table are the fastest runtimes, excluding the ones of *glo* (*glo* is the fastest in most cases).

n	$\forall$ *iglo* $\forall$	$\forall$ *iglo*	*iglo* $\forall$	gII	Fails	*glo*	Fails
8	**0.02**	**0.02**	**0.02**	**0.02**	256	0.01	256
9	**0.07**	**0.07**	**0.07**	**0.07**	925	0.06	929
10	0.29	**0.25**	0.26	0.26	4066	0.23	4106
11	1.27	**1.12**	1.13	1.13	17393	0.99	17601
12	6.06	5.36	**5.34**	5.38	78974	4.74	80011
13	30.43	27.11	**27.08**	27.27	386437	23.82	392128
14	164.63	146.29	**145.84**	146.58	2066779	128.2	2101047
15	957.12	851.15	846.3	**845.44**	11517753	741.85	11724826
16	5940.42	5223.65	5246.9	**5222.29**	69348242	4593.78	70692998

Table 7.6: Result for int-int channeling between models $\mathbf{Q_c}$ and Q_r of the N-Queens Problem

Results in Tables 7.6, 7.7, 7.8, 7.9, and 7.10 show that $\forall$ *iglo*, *iglo* $\forall$ and our gII implementations perform similarly, and it is unclear which is better in different situations. The reason for the different runtime between the implementation of $\forall$ *iglo* and *iglo* $\forall$ is the order of constraint propagation, as their global `allDiff` constraints $\forall$ are posted on different models. Moreover, the *glo* implementation is the fastest in most cases, except the one of channeling models A_n and $\mathbf{A_p}$ of the All Interval Series Problem, for the cases $n \geq 12$. These exceptions are due to the large decrease in fails. For example, in Table 7.10, when $n = 17$, the fails of the *glo* implementation are 60.5% more than those of

n, k	$\forall$ *iglo* $\forall$	$\forall$ *iglo*	*iglo* $\forall$	*gII*	Fails	*glo*	Fails
7,2	0.02	**0.01**	**0.01**	**0.01**	85	0.01	93
8,2	0.06	**0.05**	**0.05**	**0.05**	332	0.04	340
9,2	0.33	**0.29**	**0.29**	0.31	2703	0.26	2800
10,2	1.73	**1.56**	**1.56**	1.63	12860	1.39	13345
11,2	13.79	**12.45**	12.5	12.79	68844	10.94	71984
12,2	88.88	**80.77**	80.81	83.1	417953	71.53	438141
7,3	**0.01**	**0.01**	**0.01**	**0.01**	22	0.01	22
8,3	**0.02**	**0.02**	**0.02**	**0.02**	61	0.02	61
9,3	**0.09**	**0.09**	**0.09**	**0.09**	245	0.09	257
10,3	0.35	**0.32**	**0.32**	0.35	756	0.31	788
11,3	1.41	**1.3**	1.33	1.4	2813	1.27	2977
12,3	7.33	**6.83**	6.84	7.31	12996	6.6	13687
13,3	40.81	38.46	**38.22**	40.95	65458	37.37	69376
14,3	184.46	172.84	**171.74**	184.29	265118	169.59	281728
7,4	**0.01**	**0.01**	**0.01**	**0.01**	8	0	8
8,4	**0.02**	**0.02**	**0.02**	**0.02**	23	0.02	23
9,4	**0.04**	**0.04**	**0.04**	**0.04**	43	0.04	44
10,4	0.13	0.13	**0.12**	**0.12**	129	0.12	130
11,4	0.45	**0.42**	**0.42**	0.45	406	0.41	414
12,4	1.68	1.61	**1.59**	1.76	1274	1.6	1344
13,4	6.93	6.65	**6.56**	7.23	4841	6.69	5111
14,4	26.42	25.5	**25.17**	27.8	16041	25.86	16944
15,4	102.01	98.44	**97.59**	106.8	56324	96.14	59479

Table 7.7: Result for int-int channeling between models $\mathbf{L_n}$ and L_p of the Langford's Problem

n,k	$\forall$ *iglo* $\forall$	$\forall$ *iglo*	*iglo* $\forall$	*gII*	Fails	*glo*	Fails
7,2	**0.02**	**0.02**	**0.02**	**0.02**	75	0.01	82
8,2	**0.05**	**0.05**	**0.05**	**0.05**	262	0.04	291
9,2	0.3	**0.27**	0.28	0.28	2374	0.24	2575
10,2	1.65	**1.42**	1.51	1.51	11458	1.31	12531
11,2	13.19	**11.51**	12.06	11.88	60583	10.49	67765
12,2	84.42	**74.59**	77.86	76.99	359073	68.55	405667
7,3	**0.01**	**0.01**	**0.01**	**0.01**	29	0.01	31
8,3	**0.02**	**0.02**	**0.02**	**0.02**	52	0.02	54
9,3	0.1	**0.09**	**0.09**	**0.09**	189	0.08	205
10,3	0.33	**0.29**	0.31	0.32	612	0.28	645
11,3	1.41	**1.26**	1.34	1.31	2297	1.18	2426
12,3	6.35	**5.57**	6.06	5.91	9297	5.42	9923
13,3	33.04	**29.55**	31.82	31.15	44221	28.75	47416
14,3	136.93	**121.51**	131.04	128.61	160433	120.4	173291
7,4	**0.01**	**0.01**	**0.01**	**0.01**	10	0.01	10
8,4	**0.03**	**0.03**	**0.03**	0.03	25	0.03	25
9,4	**0.07**	**0.07**	**0.07**	0.07	52	0.06	56
10,4	0.2	**0.19**	**0.19**	0.2	125	0.18	138
11,4	0.44	**0.41**	0.43	0.44	261	0.41	272
12,4	1.74	**1.65**	1.71	1.74	900	1.64	947
13,4	5.61	**5.31**	5.56	5.67	2460	5.42	2627
14,4	17.06	**16.31**	17.02	17.48	6822	16.64	7304
15,4	58.45	**55.34**	58.01	59.08	21354	54.56	22775

Table 7.8: Result for int-int channeling between models L_n and $\mathbf{L_p}$ of the Langford's Problem

n	$\forall$ *iglo* $\forall$	$\forall$ *iglo*	*iglo* $\forall$	*gII*	Fails	*glo*	Fails
8	**0.01**	**0.01**	**0.01**	**0.01**	103	0.01	104
9	0.05	**0.04**	**0.04**	**0.04**	347	0.04	349
10	0.18	**0.17**	**0.17**	**0.17**	1284	0.15	1298
11	0.81	0.73	**0.72**	0.75	5077	0.65	5136
12	3.81	3.44	**3.37**	3.53	21887	3.11	22238
13	18.91	17.05	**16.85**	17.59	99625	15.5	101463
14	101.11	90.97	**88.9**	93.4	485829	83.33	495826
15	566.92	514.76	**502.86**	524.65	2499948	472.83	2558523
16	3430.03	3083.98	**3016.11**	3144.38	13748263	2850.82	14099360

Table 7.9: Result for int-int channeling between models $\mathbf{A_n}$ and A_p of the All Interval Series Problem

n	$\forall$ *iglo* $\forall$	$\forall$ *iglo*	*iglo* $\forall$	*gII*	Fails	*glo*	Fails
9	**0.02**	**0.02**	**0.02**	**0.02**	115	0.02	120
10	0.05	**0.04**	**0.04**	**0.04**	308	0.03	324
11	0.14	0.12	0.12	**0.11**	904	0.11	981
12	0.41	0.37	0.38	**0.33**	2760	0.35	3146
13	1.39	1.24	1.31	**1.11**	9051	1.27	10892
14	5.32	4.68	4.98	**4.17**	31737	5.09	40352
15	22.39	19.98	21.24	**17.47**	126407	23.18	173549
16	104.19	92.52	99.25	**80.09**	540979	114.67	794100
17	539.06	482.85	515.93	**415.53**	2593350	643.59	4162212

Table 7.10: Result for int-int channeling between models A_n and $\mathbf{A_p}$ of the All Interval Series Problem

implementations maintaining GAC. Thus, implementations maintaining GAC on int-int channeling perform better, if they can cause much more domain reduction than the *glo* implementation.

7.2 Set-Int Channeling Constraint

Tables 7.11, 7.12, 7.13, 7.14, 7.15, and 7.16 give the results of set-int channeling between models $\mathbf{G_p}$ and G_g, G_p and $\mathbf{G_g}$, $\mathbf{G_w}$ and G_g, G_w and $\mathbf{G_g}$, $\mathbf{B_p}$ and B_c, and B_p and $\mathbf{B_c}$ respectively. In addition to the standard *iff* and *ele* implementations, we also have $\prod$ *iff*, which is *iff* augmented with the set partition constraints $\prod$. We prove that keeping the partition constraints in the set model does not increase pruning in Chapter 5. We use the $\prod$ *iff* implementation to study how much the partition constraints degrade performances. For the realization of the set partition constraints, we use the `IlcPartition` constraint, which is a predefined constraint in ILOG Solver.

Results in Tables 7.11, 7.12, 7.13, 7.14, 7.15, and 7.16 confirm that *glo* is the fastest among all other implementations. The speedups for the $\prod$ *iff* implementation range from 1.17 to 1.48. One may argue that the speedup is not significant,but this will be discussed in a later section. The $\prod$ *iff* implementation are always the slowest, but with some exceptional cases in which the *iff* implementation can be a little bit slower. The reason is the same as why $\neq$ *iff* $\neq$ can be faster than *iff*. The `IlcPartition` constraint can efficiently reduce the number of propagations over the "inefficient" *iff*, though it does not increase any domain reduction. This is also the reason why the performance of the $\prod$ *iff* and *iff* implementations are similar.

g, s, w	$\prod$ *iff*	*iff*	*ele*	*glo*	Fails
3,2,2	0.01	0.01	0.01	0.01	0
3,2,3	0.01	0.01	0.01	0.01	1
3,2,4	0.01	0.01	0.01	0.01	3
3,2,5	0.01	0.01	0.01	0.01	1
4,2,2	0.01	0.01	0.01	0.01	0
4,2,3	0.03	0.03	0.03	0.02	22
4,2,4	0.11	0.1	0.09	0.09	66
4,2,5	0.15	0.14	0.13	0.12	62
4,3,2	0.01	0.02	0.01	0.01	0
4,3,3	0.12	0.1	0.08	0.08	285
4,3,4	0.2	0.19	0.16	0.15	621
4,3,5	0.16	0.15	0.11	0.11	381
5,2,2	0.02	0.02	0.02	0.02	0
5,2,3	1.28	1.23	0.96	0.91	1090
5,2,4	47.88	46.27	37.07	35.22	52702
5,2,5	514.95	498.34	407.9	389.82	629518
5,3,2	0.06	0.05	0.04	0.04	0
5,3,3	245.03	238.05	176.18	163.41	434115
5,4,2	0.05	0.04	0.03	0.03	0
5,4,3	189.66	185.4	132.3	121.66	544314
5,4,4	2220.36	2169.49	1606.71	1492.05	7908227
5,4,5	2306.79	2243.35	1674.41	1564.17	6402199
6,2,2	0.09	0.09	0.07	0.07	0
6,2,3	105.26	102.25	80.06	77.93	67595
6,3,2	1.32	1.28	0.94	0.85	0
6,4,2	1.19	1.17	0.8	0.72	0
6,5,2	0.17	0.14	0.13	0.13	0
7,2,2	0.7	0.66	0.49	0.46	0
7,3,2	66.61	64.63	44.89	41.02	0
7,4,2	281.68	276.87	187.94	170.92	0
7,5,2	52.75	51.68	34.57	31.37	0
7,6,2	0.59	0.61	0.49	0.41	0
Speedup	1.52(0.12)	1.48(0.12)	1.08(0.03)	1(0)	13

Table 7.11: Result for set-int channeling between models $\mathbf{G_p}$ and G_g of the Social Golfer Problem

g, s, w	$\prod$ *iff*	*iff*	*ele*	*glo*	Fails
3,2,2	0.01	0.01	0.01	0.01	0
3,2,3	0.01	0.01	0.01	0.01	1
3,2,4	0.01	0.01	0.01	0.01	3
3,2,5	0.01	0.01	0.01	0.01	1
4,2,2	0.01	0.01	0.01	0	0
4,2,3	0.02	0.03	0.03	0.02	14
4,2,4	0.1	0.09	0.09	0.08	60
4,2,5	0.13	0.13	0.12	0.1	80
4,3,2	0.02	0.01	0.01	0.01	0
4,3,3	0.09	0.07	0.07	0.06	84
4,3,4	0.15	0.16	0.13	0.11	341
4,3,5	0.05	0.05	0.05	0.04	55
5,2,2	0.02	0.02	0.02	0.01	0
5,2,3	1.1	1.05	0.85	0.8	526
5,2,4	38.63	37.24	31.01	29.55	19696
5,2,5	412.4	397.43	34336.56	322.82	251678
5,3,2	0.05	0.04	0.04	0.04	0
5,3,3	183.31	176.9	138.38	130.31	151569
5,4,2	0.05	0.05	0.05	0.05	4
5,4,3	88.94	87.33	66.44	61.88	106224
5,4,4	1363.52	1346.27	1008.05	937.94	2508285
5,4,5	659.67	653.66	489.24	457.15	824135
6,2,2	0.09	0.09	0.07	0.06	0
6,2,3	87.43	84.58	67.09 63.26	29136	
6,3,2	1.45	1.32	1.1	0.98	0
6,4,2	1.49	1.53	1.09	1	362
6,5,2	0.17	0.17	0.15	0.13	65
7,2,2	0.64	0.62	0.48	0.45	0
7,3,2	76.38	73.97	54.41	50.43	168
7,4,2	404.97	398.91	288.58	266.73	60729
7,5,2	95.89	95.25	68.88	63.53	45983
7,6,2	1.17	1.2	0.87	0.84	898
Speedup	1.43(0.08)	1.4(0.09)	1.06(0.02)	1(0)	14

Table 7.12: Result for set-int channeling between models G_p and $\mathbf{G_g}$ of the Social Golfer Problem

g,s,w	$\prod$ *iff*	*iff*	*ele*	*glo*	Fails
3,2,2	0.01	0.01	0.01	0.01	0
3,2,3	0.01	0.01	0.01	0.01	2
3,2,4	0.01	0	0	0	8
4,2,2	0.01	0.01	0.01	0.01	0
4,2,3	0.31	0.28	0.26	0.25	142
4,2,4	6.3	5.97	5.15	5	4695
4,3,2	0.02	0.01	0.01	0.01	0
4,3,3	0.72	0.68	0.62	0.6	900
4,3,4	8.74	8.27	7.33	7.14	17024
5,2,2	0.06	0.06	0.05	0.05	0
5,2,3	157.69	148.95	132.26	128.99	52486
5,3,2	0.09	0.1	0.09	0.08	14
5,3,3	11004.7	10371.3	9374.17	9212.21	9712202
5,4,2	0.06	0.05	0.05	0.05	4
5,4,3	4815.62	4687.03	4204.2	4138.08	4695132
6,2,2	1.11	1.03	0.96	0.95	0
6,3,2	6.47	6.28	5.79	5.73	1020
6,4,2	3.55	3.36	3.14	3.08	1077
6,5,2	0.27	0.28	0.25	0.25	65
7,2,2	39.98	38.11	34.68	33.94	0
7,3,2	985.56	933.35	871.84	860.37	97173
7,4,2	1884.82	1820.12	1691.51	1666.95	455682
7,5,2	266.62	258.51	244.67	240.65	84423
7,6,2	2.25	2.35	2.23	2.09	898
Speedup	1.17(0.05)	1.12(0.04)	1.02(0.02)	1(0)	13

Table 7.13: Result for set-int channeling between models $\mathbf{G_w}$ and G_g of the Social Golfer Problem

g,s,w	$\prod$ *iff*	*iff*	*ele*	*glo*	Fails
3,2,2	0.01	0.01	0.01	0	0
3,2,3	0.01	0.01	0.01	0.01	2
3,2,4	0.01	0.01	0.01	0.01	5
4,2,2	0.01	0.01	0.01	0.01	0
4,2,3	0.27	0.26	0.23	0.22	164
4,2,4	5.12	4.8	4.12	3.99	3985
4,3,2	0.02	0.02	0.02	0.02	0
4,3,3	0.6	0.57	0.52	0.49	504
4,3,4	5.02	4.76	4.07	3.98	10207
5,2,2	0.05	0.05	0.05	0.05	0
5,2,3	140.57	131.78	116.5	113.61	60187
5,3,2	0.1	0.09	0.08	0.07	6
5,3,3	8563.76	8098.96	7326.74	7190.13	4939024
5,4,2	0.05	0.03	0.05	0.04	4
5,4,3	3206.92	3049.01	2758.35	2720.2	2549284
6,2,2	1.09	0.96	0.89	0.88	0
6,3,2	5.95	5.71	5.34	5.27	338
6,4,2	3.35	3.2	3	2.95	780
6,5,2	0.28	0.27	0.26	0.26	65
7,2,2	36.76	34.72	31.5	30.76	0
7,3,2	896.59	844.56	785.67	777.37	30443
7,4,2	1694.84	1639.84	1528.36	1508.1	249735
7,5,2	252.19	245.49	230.9	225.95	66902
7,6,2	2.4	2.45	2.12	1.99	898
Speedup	1.18(0.06)	1.13(0.05)	1.02(0.02)	1(0)	12

Table 7.14: Result for set-int channeling between models G_w and $\mathbf{G_g}$ of the Social Golfer Problem

instance	$\prod$ *iff*	*iff*	*ele*	*glo*	Fails
8 periods	0.09	0.08	0.06	0.05	101
10 periods	0.63	0.61	0.46	0.45	470
12 periods	44.74	44.31	30.62	29.32	33530
Speedup	1.46(0.09)	1.43(0.11)	1.03(0.02)	1(0)	2

Table 7.15: Result for set-int channeling between models $\mathbf{B_p}$ and B_c of the Balanced Academic Curriculum Problem

instance	$\prod$ *iff*	*iff*	*ele*	*glo*	Fails
8 periods	0.98	0.94	0.78	0.69	1577
10 periods	0.33	0.3	0.24	0.23	323
12 periods	1.66	1.65	1.3	1.23	882
Speedup	1.38(0.05)	1.35(0.01)	1.09(0.05)	1(0)	2

Table 7.16: Result for set-int channeling between models B_p and $\mathbf{B_c}$ of the Balanced Academic Curriculum Problem

7.3 Set-Set Channeling Constraint

Tables 7.17, 7.18, 7.19, and 7.20 give the results of set-set channeling between models $\mathbf{G_p}$ and G_w, G_p and $\mathbf{G_w}$, $\mathbf{S_n}$ and S_p, and S_n and $\mathbf{S_p}$ respectively. Result confirms that *glo* is the fastest among all implementations. The speedups for the *iff* implementation range from 1.27 to 1.36. Again reasons on influencing the speedup will be discussed in a later section.

g, s, w	*iff*	*ele*	*glo*	Fails
3,2,2	0	0	0	2
3,2,3	0.01	0.01	0.01	13
4,2,2	0.01	0.01	0.01	8
4,2,3	0.21	0.17	0.17	229
4,3,2	0.35	0.28	0.27	938
4,3,3	20	15.98	15.14	45344
5,2,2	0.11	0.09	0.08	72
5,2,3	17.88	14.43	13.76	13561
5,3,2	35.95	27.88	26.09	63389
5,4,2	4102.68	3074.57	2851.63	10754086
6,2,2	1.21	0.97	0.92	688
6,3,2	5534.17	4207.29	3932.27	7656122
7,2,2	16.97	13.4	12.66	8272
Speedup	1.36(0.05)	1.06(0.02)	1(0)	7

Table 7.17: Result for set-set channeling between models $\mathbf{G_p}$ and G_w of the Social Golfer Problem

7.4 Int-Bool Channeling Constraint

Tables 7.21, 7.22, 7.23, 7.24, 7.25, 7.26, and 7.27 give the results of set-set channeling between models Q_c and Q_z, L_n and L_z, L_p and L_z, A_n and A_z,

g, s, w	*iff*	*ele*	*glo*	Fails
3,2,2	0.01	0.01	0.01	2
3,2,3	0.01	0.01	0.01	18
4,2,2	0.01	0.01	0.01	8
4,2,3	0.25	0.22	0.2	607
4,3,2	0.32	0.27	0.25	684
4,3,3	23.86	19.81	18.81	77635
5,2,2	0.1	0.09	0.08	60
5,2,3	20.29	16.97	16.23	36744
5,3,2	32.39	26.78	25.38	47988
5,4,2	3029.6	2512.79	2373.13	5764608
6,2,2	1.11	0.92	0.87	544
6,3,2	5159.86	4183.58	3963.61	5498928
7,2,2	15.47	12.86	12.21	6040
Speedup	1.27(0.02)	1.05(0.01)	1(0)	7

Table 7.18: Result for set-set channeling between models G_p and $\mathbf{G_w}$ of the Social Golfer Problem

n	*iff*	*ele*	*glo*	Fails
9	0.83	0.68	0.65	3786
10	50.59	40.76	38.77	179583
12	498684	399545	379782	1073741849
13	613560	490844	468035	1073741851
Seedup	1.31(0)	1.06(0.01)	1(0)	3

Table 7.19: Result for set-set channeling between models $\mathbf{S_n}$ and S_p of the Steiner Triple Systems

A_p and A_z, G_g and G_z, and B_c and B_z respectively. Each table is separated into table (a) and (b), which are the results by choosing search variables in the first and the second model respectively. In addition to the standard *iff* implementations, we also have *iff* $\odot$, which is *iff* augmented with the sum-to-one constraint $\odot$. There is also our *ele* implementation, but we find that its performances are *basically identical* to *glo*, thus we leave out the results.

We prove that keeping the sum-to-one constraint in the Boolean model does not increase pruning in Chapter 5. We use the *iff* $\odot$ implementation to study how much the sum-to-one constraint degrade performances. For the realization of the sum-to-one constraint, we use the `IlcSum` constraint, which is a predefined constraint in ILOG Solver.

Results in Tables 7.21, 7.22, 7.23, 7.24, 7.25, 7.26, and 7.27 confirm that *glo* is the fastest among all implementations. The speedups for the *iff* $\odot$ implementation range from 1.04 to 3.33. Again reasons on influencing the speedup will be discussed in a later section. The *iff* $\odot$ implementation is always the slowest, but with some exceptional cases in which the *iff* implementation can be a little bit slower. The reason is the same as why $\neq$ *iff* $\neq$ or $\prod$ *iff* can be faster than *iff*. The `IlcSum` constraint can efficiently reduce the number of propagation steps over the "inefficient" *iff*, though it does not increase any domain reduction. This is also the reason why the performance of the *iff* $\odot$ and *iff* implementations are similar.

7.5 Set-Bool Channeling Constraint

Tables 7.28, 7.29, 7.30, 7.31, and 7.32 give the results of set-bool channeling between models G_p and G_z, G_w and G_z, B_p and B_z, S_n and S_z, and S_p and S_z respectively. Each table is separated into table (a) and (b), which are the

n	*iff*	*ele*	*glo*	Fails
9	1.33	1.03	0.98	6362
10	21.73	18.11	17.27	107532
12	296684	247781	237486	1073741848
13	583889	486361	467110	1073739057
Speedup	1.28(0.05)	1.04(0.01)	1(0)	4

Table 7.20: Result for set-set channeling between models S_n and $\mathbf{S_p}$ of the Steiner Triple Systems

n	*iff* $\odot$	*iff*	*glo*	Fails
7	0.01	0.01	0.01	62
8	0.03	0.02	0.02	256
9	0.11	0.11	0.06	929
10	0.46	0.45	0.23	4106
11	2.09	2.06	1	17601
12	10.47	10.37	4.76	80011
13	56.53	55.24	24.47	392128
14	318.33	311.19	134.73	2101047
15	1932.62	1885.62	780.01	11724826
16	12580.8	12269.5	4912.93	70692998
Speedup	2.33(0.17)	2.29(0.16)	1(0)	6

(a) Search by X_c

n	*iff* $\odot$	*iff*	*glo*	Fails
7	0.01	0.01	0	65
8	0.03	0.03	0.01	300
9	0.11	0.11	0.06	1151
10	0.45	0.46	0.26	5181
11	2.18	2.19	1.16	23515
12	11.35	11.38	5.77	111076
13	62.29	62.38	30.83	561362
14	363.54	364.36	172.69	3079792
15	2280.16	2272.72	1047.69	17692260
16	15113.9	15109.8	6709.7	109047332
Speedup	2.07(0.14)	2.07(0.13)	1(0)	6

(b) Search by X_z

Table 7.21: Result for int-bool channeling between models Q_c and Q_z of the N-Queens Problem

n,k	*iff* ⊙	*iff*	*glo*	Fails
7,2	0.04	0.04	0.02	110
8,2	0.15	0.14	0.06	368
9,2	1	0.96	0.38	3211
10,2	5.8	5.56	2.19	15597
11,2	47.35	45.02	17.09	91471
12,2	315.43	300.35	110.03	557590
7,3	0.03	0.03	0.01	28
8,3	0.08	0.08	0.03	75
9,3	0.35	0.33	0.13	313
10,3	1.39	1.35	0.48	1064
11,3	6.33	6.02	2.1	4425
12,3	33.47	32.09	10.83	20273
13,3	201.97	194.82	63.05	107233
14,3	941.78	904.63	292.31	439230
7,4	0.02	0.03	0	10
8,4	0.07	0.07	0.02	28
9,4	0.19	0.17	0.07	62
10,4	0.58	0.55	0.19	165
11,4	2.39	2.22	0.69	635
12,4	9.92	8.98	2.62	2144
13,4	49.08	44.65	11.5	8558
14,4	199.46	178.55	42.67	28787
Speedup	3.33(0.62)	3.12(0.5)	1(0)	12

(a) Search by X_n

n,k	*iff* ⊙	*iff*	*glo*	Fails
7,2	0.04	0.03	0.02	116
8,2	0.14	0.13	0.07	466
9,2	0.88	0.84	0.35	3453
10,2	4.96	4.72	2.04	17194
11,2	38.3	36.35	15.32	98505
12,2	248.03	236.04	96.14	606013
7,3	0.03	0.03	0.01	32
8,3	0.07	0.07	0.03	71
9,3	0.27	0.26	0.1	243
10,3	0.9	0.86	0.33	741
11,3	3.52	3.42	1.3	2757
12,3	16.11	15.58	5.66	11336
13,3	78.06	76	26.49	48960
14,3	343.97	338.62	116.14	197640
7,4	0.03	0.02	0.01	10
8,4	0.08	0.08	0.04	38
9,4	0.19	0.19	0.07	71
10,4	0.48	0.45	0.17	141
11,4	1.57	1.48	0.48	392
12,4	5.07	4.53	1.33	1057
13,4	16.27	16.33	3.98	2813
14,4	56.72	52.82	12.71	8388
Speedup	3.15(0.68)	3.01(0.64)	1(0)	11

(b) Search by X_z

Table 7.22: Result for int-bool channeling between models L_n and L_z of the Langford's Problem

results by choosing search variables in the first and the second model respectively. There is also our *ele* implementation, but we find that its performances are *basically identical* to *glo*, thus we leave out the results. Result confirms that *glo* is the fastest among all other implementations. The speedups for the *iff* implementation range from 1.03 to 1.26. Reasons on influencing the speedup will be discussed in the next section.

7.6 Discussion

One might observe discrepancies in performance comparison from the theoretical prediction given in Table 6.1. For example, *glo* performs better than

n,k	*iff* $\odot$	*iff*	*glo*	Fails
7,2	0.04	0.03	0.02	104
8,2	0.16	0.16	0.08	381
9,2	1.01	1.01	0.51	3029
10,2	6.05	5.81	2.99	15318
11,2	49.53	47.99	24.28	91986
12,2	331.97	323.15	161.89	571667
7,3	0.07	0.06	0.03	124
8,3	0.22	0.21	0.1	320
9,3	1.1	1.07	0.53	1406
10,3	4.11	3.99	1.95	4748
11,3	19.83	19.25	9.42	19902
12,3	115.27	111.37	53.88	99421
13,3	778.46	760.24	356.26	597804
14,3	4880.36	4598.26	2076.42	3017268
7,4	0.06	0.05	0.02	38
8,4	0.25	0.23	0.12	175
9,4	1.16	1.1	0.54	708
10,4	3.61	3.46	1.58	1819
11,4	20.22	19.05	7.83	8120
12,4	99.25	87.49	33.39	30763
13,4	565.25	493.59	181.68	145950
14,4	2810.35	2765.93	892.19	610426
Speedup	2.33(0.4)	2.22(0.33)	1(0)	16

(a) Search by X_p

n,k	*iff* $\odot$	*iff*	*glo*	Fails
7,2	0.04	0.04	0.02	124
8,2	0.17	0.17	0.1	496
9,2	1.16	1.14	0.62	3668
10,2	6.74	6.61	3.58	18226
11,2	52.69	52.51	27.92	105202
12,2	349.06	351.89	184.14	646472
7,3	0.04	0.05	0.03	126
8,3	0.12	0.13	0.07	278
9,3	0.46	0.49	0.25	975
10,3	1.52	1.6	0.83	2757
11,3	5.98	6.49	3.28	9579
12,3	27.52	28.92	14.64	35845
13,3	133.78	144.5	70.11	145425
14,3	626.06	649.59	308.21	535418
7,4	0.03	0.02	0.03	45
8,4	0.16	0.14	0.08	285
9,4	0.37	0.39	0.19	597
10,4	1.14	1.14	0.5	1547
11,4	4.04	4.23	1.67	4557
12,4	14.73	15.59	5.39	12996
13,4	47.89	51.56	19.7	37955
14,4	202.96	200.81	71.61	116245
Speedup	2.12(0.35)	2.19(0.37)	1(0)	14

(b) Search by X_z

Table 7.23: Result for int-bool channeling between models L_p and L_z of the Langford's Problem

predicted in IB, but less in SI. There are other factors than just the type of the channeling that affect the constraint solving efficiency in real problems (instead of quasi-empty models with only channeling constraints). First, we observe that the speedups of *glo* over others grow with instance size in general. We employ all-solution search in our experiments so that the results are less sensitive to search heuristics and to exercise the channeling constraints more fully, but all-solution search is costly and limits our attentions to smaller instances. We did perform some experiments on single-solution search on larger instances. For example, in set-set channeling, for the model pair $S_n, \mathbf{S_p}$, the speedups against *iff* become 1.84 and 2.16 for $n = 25$ and $n = 27$ respectively, where n is the total number of distinct integers that can be contained

n	*iff* $\odot$	*iff*	*glo*	Fails
6	0.01	0.01	0	17
7	0.01	0.01	0.01	61
8	0.05	0.05	0.05	194
9	0.19	0.19	0.15	584
10	0.73	0.71	0.6	1900
11	3.01	2.93	2.49	6726
12	12.95	12.76	10.87	25572
13	59.36	58.77	49.7	103662
14	288.81	286.53	244.17	447656
15	1496.01	1455.34	1238.85	2034574
16	8264.58	8009.53	6753.43	9860668
Speedup	1.2(0.01)	1.18(0.01)	1(0)	6

(a) Search by X_n

n	*iff* $\odot$	*iff*	*glo*	Fails
6	0	0	0	2
7	0	0	0	16
8	0.02	0.02	0.01	67
9	0.07	0.07	0.05	255
10	0.28	0.27	0.26	1070
11	1.44	1.41	1.25	4717
12	8.02	7.84	7.02	22849
13	48.28	46.98	42.37	121632
14	299.68	293.88	264.75	652856
15	2011.51	1950.13	1779.49	3802562
16	14486.3	14125.9	12617.1	23829086
Speedup	1.14(0.01)	1.11(0.01)	1(0)	6

(b) Search by X_z

Table 7.24: Result for int-bool channeling between models A_n and A_z of the All Interval Series Problem

in each triple. Another example is SI channeling, for the model pair $\mathbf{G_p}, G_g$, the speedups against *iff* are 1.89 and 1.96 for $p = 13, g = 13, w = 3$ and $p = 14, g = 14, w = 3$ respectively, where p is the number of golfers in each group, g is the number of groups in each week, and w is the number of weeks need to be scheduled. We observe similar increase in speedup in other problems.

Second, the proportion of channeling constraints among all constraints in the model and the complexity of the other constraints also affect the results. In general, if a model contain a large proportion of complex constraints, then the speedup gained in the improved channeling constraint implementation can be insignificant as compared to the time required for solving the other constraints. For example, in model G_w, there are $O(p^2g^3)$ constraints to ensure that any two groups in different weeks have at most one golfer in common. For the combined models of $G_w andG_g$ using SI and G_p, G_w using SS, *iff* has only pgw constraints, and *ele* has one less dimension when compared with *iff*. Another example is on S_n and S_p. There are $O(n^4)$ constraints to ensure that any two triples have at most one common integer, while *iff* has only nm constraints

n	*iff* ⊙	*iff*	*glo*	Fails
6	0	0	0	1
7	0	0	0	8
8	0.01	0.01	0.01	41
9	0.04	0.04	0.03	112
10	0.12	0.12	0.1	297
11	0.37	0.36	0.31	856
12	1.19	1.16	1	2597
13	4.03	3.96	3.39	7971
14	15.01	14.89	12.67	26152
15	59.94	59.53	50.54	97205
16	276.81	271.37	232.76	387419
Speedup	1.19(0)	1.17(0.01)	1(0)	5

(a) Search by X_p

n	*iff* ⊙	*iff*	*glo*	Fails
6	0	0	0	2
7	0	0	0	16
8	0.02	0.01	0.01	74
9	0.07	0.07	0.07	282
10	0.37	0.36	0.31	1214
11	1.95	1.87	1.62	5696
12	11.07	10.78	9.32	27700
13	67.47	65.91	56.78	145751
14	433.67	424.66	361.94	817882
15	2929.05	2834.72	2440.05	4791761
16	21517.7	20636	17863	29845002
Speedup	1.2(0.01)	1.16(0.01)	1(0)	6

(b) Search by X_z

Table 7.25: Result for int-bool channeling between models A_p and A_z of the All Interval Series Problem

for combining models S_n and S_p, where $m = n(n-1)/6$.

g,s,w	*iff* ⊙	*iff*	*glo*	Fails
3,2,2	0	0.01	0	0
3,2,3	0	0.01	0	3
3,2,4	0.01	0.01	0.01	7
3,2,5	0.01	0.01	0.01	4
4,2,2	0.01	0.01	0.01	2
4,2,3	0.04	0.05	0.05	43
4,2,4	0.18	0.19	0.17	286
4,2,5	0.35	0.38	0.36	908
4,3,2	0.01	0.02	0.01	0
4,3,3	0.14	0.13	0.13	159
4,3,4	0.36	0.37	0.34	878
4,3,5	0.36	0.37	0.35	814
5,2,2	0.03	0.03	0.03	10
5,2,3	1.76	1.76	1.65	1013
5,2,4	60.54	59.38	56.4	39296
5,2,5	662.23	664.04	627.85	701208
5,3,2	0.08	0.09	0.08	3
5,3,3	308.78	306.19	292.01	284440
5,4,2	0.06	0.06	0.06	4
5,4,3	166.33	168.83	162.31	178716
5,4,4	2206.17	2209.14	2041.4	2838369
5,4,5	2781.66	2774.68	2573.88	3257943
6,2,2	0.16	0.16	0.16	44
6,2,3	135.26	133.92	125.44	53239
6,3,2	2.93	2.97	2.82	36
6,4,2	2.97	2.94	2.79	537
6,5,2	0.24	0.27	0.23	65
7,2,2	1.36	1.33	1.26	234
7,3,2	157.8	156.15	147.31	555
7,4,2	838.47	844.73	792.58	108702
7,5,2	214.58	197.28	199.21	58606
7,6,2	2.19	2.15	2.04	898
Speedup	1.07(0.02)	1.06(0.02)	1(0)	12

(a) Search by X_g

g,s,w	*iff* ⊙	*iff*	*glo*	Fails
3,2,2	0.01	0.01	0	0
3,2,3	0	0	0.01	5
3,2,4	0.01	0.01	0	9
3,2,5	0.01	0	0.01	4
4,2,2	0	0.01	0.01	4
4,2,3	0.07	0.07	0.07	176
4,2,4	0.31	0.32	0.31	971
4,2,5	0.62	0.62	0.61	2100
4,3,2	0.01	0.02	0.01	0
4,3,3	0.27	0.27	0.25	800
4,3,4	0.75	0.75	0.72	3006
4,3,5	0.48	0.47	0.47	1585
5,2,2	0.03	0.04	0.04	50
5,2,3	3.16	3.18	3.04	7285
5,2,4	138.1	138.15	135.46	367404
5,2,5	1864.72	1886.39	1865.17	5599996
5,3,2	0.1	0.1	0.1	45
5,3,3	596.45	603.05	562.55	1469974
5,4,2	0.05	0.05	0.04	0
5,4,3	509.17	502.75	474.09	1279148
6,2,2	0.25	0.25	0.24	312
6,2,3	240.88	242.37	228.42	484732
6,3,2	3.33	3.33	3.16	2691
6,4,2	2.51	2.48	2.43	816
6,5,2	0.19	0.17	0.18	0
7,2,2	2.09	2.11	1.97	2658
7,3,2	166.47	168.24	157.76	127191
7,4,2	659.2	642.74	621.79	284324
7,5,2	118.62	118.87	109.12	22650
7,6,2	0.99	1.06	1.02	0
Speedup	1.04(0.03)	1.05(0.02)	1(0)	13

(b) Search by X_z

Table 7.26: Result for int-bool channeling between models G_g and G_z of the Social Golfer Problem

instance	*iff* ⊙	*iff*	*glo*	Fails
8 periods	0.02	0.03	0.01	101
10 periods	0.18	0.17	0.16	468
12 periods	7.39	7.34	5.84	33602
Speedup	1.2(0.1)	1.16(0.14)	1(0)	2

(a) Search by X_c

instance	*iff* ⊙	*iff*	*glo*	Fails
8 periods	0.03	0.03	0.02	183
10 periods	0.15	0.15	0.13	1103
12 periods	0.11	0.1	0.08	366
Speedup	1.26(0.16)	1.2(0.07)	1(0)	2

(b) Search by X_z

Table 7.27: Result for int-bool channeling between models B_c and B_z of the Balanced Academic Curriculum Problem

g,s,w	*iff*	*glo*	Fails
3,2,2	0	0	4
3,2,3	0.01	0.01	5
3,2,4	0.01	0.01	7
3,2,5	0.01	0.01	7
4,2,2	0.02	0.01	12
4,2,3	0.05	0.05	34
4,2,4	0.17	0.15	102
4,2,5	0.29	0.26	234
4,3,2	0.02	0.02	45
4,3,3	0.18	0.16	330
4,3,4	0.33	0.27	595
4,3,5	0.33	0.27	510
5,2,2	0.05	0.03	62
5,2,3	2.12	1.83	1148
5,2,4	74.87	65.46	48468
5,2,5	769.06	680.53	544677
5,3,2	0.24	0.2	647
5,3,3	368.05	300.63	434408
5,4,2	0.18	0.17	515
5,4,3	298.13	238.99	544829
5,4,4	3239.6	2676.21	6735600
5,4,5	3593.71	2988.32	5389126
6,2,2	0.24	0.21	359
6,2,3	162.35	137.23	67254
6,3,2	6.66	5.45	17311
6,4,2	16.11	12.92	43036
6,5,2	3.01	2.38	7030
7,2,2	1.91	1.65	2682
7,3,2	304.66	247.43	677196
7,4,2	2826.54	2218.26	6684046
7,5,2	1988.17	1542.39	4060581
7,6,2	67.12	51.39	117608
Speedup	1.22(0.05)	1(0)	16

(a) Search by X_p

g,s,w	*iff*	*glo*	Fails
3,2,2	0	0	4
3,2,3	0.01	0	5
3,2,4	0.01	0.01	7
3,2,5	0.01	0.01	7
4,2,2	0.01	0.01	12
4,2,3	0.05	0.05	34
4,2,4	0.18	0.15	102
4,2,5	0.29	0.26	234
4,3,2	0.03	0.03	45
4,3,3	0.17	0.16	330
4,3,4	0.31	0.26	595
4,3,5	0.32	0.27	510
5,2,2	0.04	0.04	62
5,2,3	2.11	1.88	1148
5,2,4	75.43	67.72	48468
5,2,5	778.01	705.48	544677
5,3,2	0.24	0.19	647
5,3,3	369.02	309.26	434408
5,4,2	0.2	0.17	515
5,4,3	299.15	243.4	544829
5,4,4	3225.13	2700.2	6735600
5,4,5	3587.38	3035.35	5389126
6,2,2	0.23	0.2	359
6,2,3	163.47	142.77	67254
6,3,2	6.73	5.69	17311
6,4,2	16.24	13.39	43036
6,5,2	3.05	2.52	7030
7,2,2	1.94	1.7	2682
7,3,2	311.08	260.56	677196
7,4,2	2871.41	2321.3	6684046
7,5,2	2042.92	1609.95	4060581
7,6,2	68.78	53.57	117608
Speedup	1.19(0.05)	1(0)	16

(a) Search by X_z

Table 7.28: Result for set-bool channeling between models G_p and G_z of the Social Golfer Problem

g, s, w	*iff*	*glo*	Fails
3,2,2	0.01	0.01	4
3,2,3	0.01	0.01	10
3,2,4	0.01	0.01	9
3,2,5	0.01	0.01	9
4,2,2	0.02	0.02	20
4,2,3	0.11	0.1	227
4,2,4	0.45	0.41	892
4,2,5	0.63	0.59	1169
4,3,2	0.03	0.03	51
4,3,3	1.21	1.14	3406
4,3,4	3.63	3.41	9220
4,3,5	0.23	0.21	363
5,2,2	0.05	0.05	103
5,2,3	4.73	4.51	9576
5,2,4	164.16	153.07	291104
5,2,5	1894.42	1735.48	3139532
5,3,2	0.43	0.43	945
5,3,3	2040.06	1933.89	4591584
5,4,2	0.46	0.43	668
5,4,3	6047.96	5781.57	10227177
6,2,2	0.38	0.36	675
6,2,3	358.52	342	638346
6,3,2	15.81	15.57	29342
6,4,2	50.63	49.45	66416
6,5,2	12.41	12.4	10685
7,2,2	3.2	3.11	5287
7,3,2	814.24	794.29	1258840
7,4,2	11866.9	11792	12721242
7,5,2	10022.6	9901.37	6897490
7,6,2	447.31	441.85	215155
Speedup	1.04(0.03)	1(0)	16

(a) Search by X_w

g, s, w	*iff*	*glo*	Fails
3,2,2	0	0	4
3,2,3	0.01	0	9
3,2,4	0.01	0.01	13
3,2,5	0.01	0.01	11
4,2,2	0.02	0.01	21
4,2,3	0.09	0.08	194
4,2,4	0.42	0.41	1118
4,2,5	0.91	0.85	2596
4,3,2	0.02	0.02	45
4,3,3	0.35	0.33	845
4,3,4	1.01	0.95	3057
4,3,5	0.75	0.7	1970
5,2,2	0.05	0.05	122
5,2,3	3.9	3.71	7357
5,2,4	175.09	164.85	376394
5,2,5	2478.82	2307.03	5903049
5,3,2	0.35	0.35	907
5,3,3	753.7	729.41	1470836
5,4,2	0.24	0.22	515
5,4,3	663.91	640.01	1279663
6,2,2	0.4	0.39	780
6,2,3	293.34	281.06	487356
6,3,2	10.6	10.42	23745
6,4,2	28.49	28.06	57479
6,5,2	4.76	4.53	7030
7,2,2	3.27	3.19	5964
7,3,2	488.93	486.3	935808
7,4,2	5018.04	4977.8	8439295
7,5,2	4014.09	4006.03	5388120
7,6,2	114.94	114.04	117608
Speedup	1.03(0.02)	1(0)	14

(b) Search by X_z

Table 7.29: Result for set-bool channeling between models G_w and G_z of the Social Golfer Problem

instance	*iff*	*glo*	Fails
8 periods	6.14	4.83	15107
10 periods	0.24	0.19	328
12 periods	7.82	6.38	7092
Speedup	1.25(0.03)	1(0)	2

(a) Search by X_p

instance	*iff*	*glo*	Fails
8 periods	10470.5	8829.29	22182175
10 periods	34030.6	30523.5	66673689
12 periods	71.74	64.35	43347
Speedup	1.14(0.04)	1(0)	3

(b) Search by X_z

Table 7.30: Result for set-bool channeling between models B_p and B_z of the Balanced Academic Curriculum Problem

n	*iff*	*glo*	Fails
7	0.02	0.01	63
9	3.45	3.09	15711
10	164.01	147.8	544085
Speedup	1.11(0)	1(0)	2

(a) Search by X_n

n	*iff*	*glo*	Fails
7	0.01	0.01	63
9	4.14	3.3	15711
10	198.2	157.78	544085
Speedup	1.26(0)	1(0)	2

(b) Search by X_z

Table 7.31: Result for set-bool channeling between models S_n and S_z of the Steiner Triple Systems

instance	*iff*	*glo*	Fails
7	0.01	0	11
9	5.39	4.87	15176
10	154.21	138.15	375223
Speedup	1.11(0.01)	1(0)	2

(a) Search by X_p

instance	*iff*	*glo*	Fails
7	0.01	0.01	9
9	1.55	1.39	3786
10	91.93	82.82	179583
Speedup	1.11(0)	1(0)	2

(b) Search by X_z

Table 7.32: Result for set-bool channeling between models S_p and S_z of the Steiner Triple Systems

Chapter 8

Related Work

In this chapter, we give an overview of related work on channeling constraints. This chapter is separated into four sections: empirical studies, theoretical studies, applications, and other kinds of channeling constraints.

8.1 Empirical Studies

Cheng et al. [CCLW99] propose the concept of redundant modeling, which uses channeling constraint to combine multiply models of the same problem. They suggest guidelines and give examples on how to create models, and how to combine them by channeling constraints, and in various forms of channeling constraints. They give two cases studies, which are the n-queens problem and the nurse staff rostering problem. They use the n-queens problem to show, in detail steps, how the combined model causes extra domain reduction. The nurse staff rostering problem is a real-life problem. The combined models show significant speedup against the individual (single) models.

Smith [Smi00, Smi01] studies redundant modeling on the n-queens problem, the Langford's problem, and the social golfer problem. She points out several important issues. First, the *iff* constraints for int-int channeling can subsume the pairwise disequalities in the models, but not the global all-different constraints. Second, she proposes the concept of *minimal combined model*, in which some constraints in the combined model can be removed without affecting the search space. For example, the pairwise disequalities in the combined model by int-int channeling can be removed. Third, she suggests to re-implement the *iff* more efficiently in general, which is realized in the thesis.

8.2 Theoretical Studies

Walsh et al. [Wal01, HW02, HSW04] perform an extensive study on applying redundant modeling on permutation problems and injection problems. In other words, their study is related to int-int channeling. They define the concept of constraint tightness, which we use in our theoretical analysis, for comparing the power of domain reduction between different models of the same problem. There are two differences between their comparison and our comparison on int-int channeling. They focus on the channeling constraint in the form of *iff*, while we also study the form of *ele* and *glo*. On the other hand, they look at different local consistencies, arc consistency (AC), forward checking (FC), bounds consistency (BC), path consistency (PC), strong path consistency (ACPC), path inverse consistency (PIC), restricted path consistency (RPC), and singleton arc consistency (SAC). We study AC and GAC for integer(or Boolean) variables, set bounds consistency (SBC) for set variables, and hybrid consistent (HC) for mixed of integer and set variables.

Choi et al. [CL02, CLS06] do much further work on the idea of minimal combined model [Smi01]. They perform theoretical study on when some constraints are *propagation redundant*, which means redundant in terms of domain reduction, with respect to other constraints in the combined model. Their results are applicable to any combined model that is combined by the five kinds of channeling constraint. There are three main differences between their work and our work on channeling. First, their study involves identifying propagation redundant constraints caused by two different reasons. A constraint can be made progagation redundant by (a) the channeling constraints and/or (b) constraints in another submodel via channeling constraints. We focus on identifying propagation redundant constraint caused by channeling constraints only. Second, their study is based on the channeling form of *iff* only, while we also study the form of *ele* and *glo*. Third, their study points out that set-int channeling can subsume the all-pair null intersection constraints ($\forall i \neq j, s_i \cap s_j = \{\}$). We further point out that set-int channeling constraint can subsume the partition constraint.

8.3 Applications

Flener et al. [FFH+02a] identify row and column symmetries in 2-dimensional matrix models [FFH+01, FFH+02a]. They are variable symmetries, and can be broken by adding lexicographical ordering constraints [CB02a, CB02b, FHK+02]. One of their studies proposes to break value symmetries using Boolean model and channeling constraints. Given an n dimension matrix model, breaking its value symmetries can be done by breaking the corresponding variable symmetries in its $n+1$ dimensional Boolean matrix model, and combining them together with int-bool channeling or set-bool channeling.

Law and Lee [Law05, LL06] proposed two methods of using symmetry breaking constraints to break value symmetries in CSP. One of them uses multiple viewpoints and channeling constraints. Given a model M which is a triple (X, D, C), where X is the variables, D is the domains, and C is the constraints. We say that a *viewpoint* V is the pair of (X, D). Thus the model M can also be expressed as the pair (V, C). Given two viewpoints V_1 and V_2 of a problem, Law and Lee prove when a value symmetry in V_1

corresponds to a variable symmetry in V_2 and *vice versa*. Moreover, they establish theorems to identify when variable symmetry breaking constraints in both V_1 and V_2 connected by channeling constraints are consistent. Their theorems are applicable to the five kinds of channeling constraints.

Law and Lee [LL02, Law02] present a method to generate a new model from an existing model through channeling constraints The process is called *model induction*. Hernández and Frisch [HF05] present how to generate channeling constraints automatically. Specifically, they use an automatic modeling tool, Conjure [FJHM05], to generate CSP models from problem specifications automatically. They target on generating channeling constraints between the generated models by Conjure, so that it is possible to produce new combined models with possibly more constraint propagation.

Many permutation problems, such as Quasigroups, Golomb Rulers, and Magic Squares in CSPLIB [GW99], can be solved more efficiently by channeling their own integer models [Wal01, HW02, DdVC03b, DdVC03a]. Hnich et al. [HPS05] study a problem called t-covering array problem, show that the problem can be solved efficiently by combining its Boolean model and integer model together by int-bool channeling, and breaking the row and column symmetries in the Boolean model.

8.4 Other Kinds of Channeling Constraints

Smith [Smi01] proposes a kind of channeling constraint which is for pair-based models. Here, we refer to her example on the social golfers problem for explanation. In Model $G_q = (X_q, D_{X_q}, C_{X_q})$, each variable $q_{i,j} \in X_q$ (integer variable) represents the week which golfer i and golfer j play in the same group ($|X_q| = n \times n$). Thus $D_{q_{i,j}} = \{1, \ldots, w\}$ represents the possible weeks.

In Model $G_H = (X_H, D_{X_H}, C_{X_H})$, each variable $H_{i,j} \in X_H$ (set variable) represents the set of golfers play with golfer i in week j ($|X_H| = n \times w$). Thus $PS(H_{i,j}) = \{1, \ldots, n\}$ represents the possible golfer numbers. We can combine G_q with G_H by:

$$q_{i,j} = k \Leftrightarrow H_{i,k} = H_{j,k} \qquad \forall q_{i,j} \in X_q, \forall k \in D_{q_{i,j}}$$

and

$$q_{i,j} \neq k \Leftrightarrow H_{i,k} \cap H_{j,k} = \{\} \qquad \forall q_{i,j} \in X_q, \forall k \in D_{q_{i,j}}$$

Flener et al. [FFH+02b] propose another two kinds of channeling constraints. The first one is relating integer variables and Boolean variables. Suppose X is a set of integer variables and Y is a set of Boolean variables. They can be channeled by:

$$x_i = j \Rightarrow y_j = 1 \qquad \forall x_i \in X, \forall y_j \in Y$$

This channeling is for indicating whether there exists any variable, say x_i, is assigned with a value, say j. Result is stored at variable y_j. The second one is relating Boolean variables and Boolean variables. Suppose X is a set of Boolean variables, and y is a Boolean variable. They can be channeled by:

$$x_i = 1 \Rightarrow y = 1 \qquad \forall x_i \in X$$

This channeling is for indicating whether there exists any variable, say x_i, is assigned with value 1. Result is stored at variable y. These two kinds of channeling constraint are not for redundant modeling. They are just for transforming some information from a set of variables to another set of variables.

Chapter 9

Concluding Remarks

We conclude the thesis in this chapter by summarizing our contributions and giving possible directions for future research.

9.1 Contributions

The thesis gives a comprehensive treatise in comparing the constraint tightness of various implementations of five common channeling constraints. Table 9.1 shows a summary for all the important theorems. These results, however, must be interpreted with care. First, it may be theoretically nice to maintain tighter consistency level to prune more values, but the associated constraint propagation algorithms might incur higher costs. For example, our gII implementation, which achieves GAC on int-int channeling constraint, cannot outperform our *glo* implementation, which achieves AC on each constraint in *iff*, although it prunes the most. Second, except for the case of II, our theoretical results suggest that maintaining HC on a global constraint would not give more pruning. This should not be understood as an argument against global constraint implementations. It is always possible to implement a global constraint using a constraint propagation algorithm that maintains a lower level of consistency than HC. We have proposed two efficient propagators for

Channeling Form	Theorems
II	$GAC_{\{ii\}} = GAC_{\{\forall,ii,\forall\}}$
	$AC_{\{iff\}} = GAC_{\{ele\}}$
	$GAC_{\{ii\}} = GAC_{\{\forall,iff\}} = GAC_{\{iff,\forall\}}$
	$GAC_{\{ii\}} > AC_{\{iff\}}$
	$AC_{\{iff\}} = AC_{\{\neq,iff,\neq\}}$
SI	$HC_{\{si\}} = HC_{\{\prod,si\}}$
	$HC_{\{si\}} = HC_{\{iff\}}$
SS	$SBC_{\{ss\}} = SBC_{\{iff\}}$
IB	$GAC_{\{ib\}} = GAC_{\{ele\}} = AC_{\{iff\}}$
	$AC_{\{iff\}} = GAC_{\{iff,\odot\}}$
SB	$HC_{\{sb\}} = HC_{\{ele\}} = HC_{\{iff\}}$

Table 9.1: Summary of Theorems

implementing global channeling constraints. The `gElement` propagator is for a generalized `element` constraint, which provides "partial" globalization for the basic *iff* implementation. The *glo* propagator encapsulates all *iff* constraints into one, and achieves HC on *iff*. Experimental result confirms the efficiency of the *glo* implementation with speedups ranging from 1.0 to 3.61. While the `gElement` propagator is less efficient than the *glo* propagator, the `gElement` propagator has a speedup ranging from 1.1 to 1.4 over ILOG Solver's `element` constraint. Moreover, the *glo* implementation is on par with ILOG Solver's state of the art `IlcInverse`. Note that `IlcInverse` is specially designed for II channeling, while *glo* is a generic propagator for all five channeling constraints.

9.2 Future Work

First, in term of breath, there exists other kinds of channeling constraint, other than the five common channeling constraints we studied. For example, pair-based models needs a special form of channeling constraints, which is proposed by Smith [Smi01]. Thus, a more general channeling constraint framework can be achieved.

Second, in term of depth, more consistency level can be studied. For example, it is possible to incorporate cardinality reasoning on the channeling constraints involving set variables. Another example is about bounds consistency [MS98] on constraints with integer variables. Again, in this way, a more general channeling constraint framework can be achieved.

Third, it is interesting to study if we can optimize gII implementation further, so that it can outperform glo implementation under the int-int channeling situation.

Bibliography

[Ber70] C. Berge. *Graphe et Hypergraphes.* Dunod, Paris, 1970. [70]

[BHBHW05] Christian Bessiere, Emmanuel Hebrard, Zeynep Kiziltan Brahim Hnich, and Toby Walsh. The range and roots constraints: Specifying counting and occurrence problems. In *Proceedings of IJCAI-2005*, 2005. [9]

[CB02a] M. Carlsson and N. Beldiceanu. Arc-consistency for a chain of lexicographic ordering constraints. Technical Report T2002-18, Swedish Institute of Computer Science, 2002. [103]

[CB02b] M. Carlsson and N. Beldiceanu. Revisiting the lexicographic ordering constraint. Technical Report T2002-17, Swedish Institute of Computer Science, 2002. [103]

[CCLW99] B. M. W. Cheng, Kenneth M. F. Choi, J. H. M. Lee, and J. C. K. Wu. Increasing constraint propagation by redundant modeling: an experience report. *Constraints*, 4(2):167–192, 1999. [1, 3, 17, 30, 63, 101]

[CL02] C. W. Choi and J. H. M. Lee. On the pruning behaviour of minimal combined models for permutation csps. In

Proceedings of the International Workshop on Reformulating Constraint Satisfaction Problems: Towards Systematisation and Automation (CP2002), Cornell University, Ithaca, NY, USA, 2002. Available from http://www-users.cs.york.ac.uk/ frisch/Reformulation/02/Proceedings/. [102]

[CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, second edition, 2001. [2, 8]

[CLS00] K.M.F. Choi, J.H.M. Lee, and P.J. Stuckey. A Lagrangian reconstruction of GENET. *Artificial Intelligence*, 123:1–39, 2000. [8]

[CLS06] C. W. Choi, J. H. M. Lee, and P. J. Stuckey. Removing propagation redundant constraints in redundant modeling. *(to appear) ACM Transaction on Computational Logic*, 2006. [6, 16, 18, 19, 20, 21, 22, 23, 30, 57, 72, 102]

[COS01] COSYTEC. *CHIP 5.4, CHIP++ Reference Manual*, 2001. [2, 4, 32, 39, 41]

[DdVC03a] Iván Dotú, Álvaro del Val, and Manuel Cebrián. Channeling constraints and value ordering in the quasigroup completion problem. In *Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1372–1373, 2003. [104]

[DdVC03b] Iván Dotú, Álvaro del Val, and Manuel Cebrián. Redundant modeling for the quasigroup completion problem. In *Ninth International Conference on Principles and Practice of Constraint*

Programming (CP), volume 2833 of *LNCS*, pages 288–302, 2003. [104]

[DP87] D. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987. [2, 8]

[DSvH88] M. Dincbas, H. Simonis, and P. van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In *European Conference on Artificial Intelligence (ECAI)*, pages 290–295, 1988. [1]

[DTWZ94] A. Davenport, E. Tsang, C.J. Wang, and K. Zhu. GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proceedings of AAAI'94*, pages 325–330, 1994. [8]

[ECL05] ECLiPSe. *ECLiPSe 5.8, Constraint Library Manual*, 2005. Available from http://eclipse.crosscoreop.com/eclipse/doc/libman/libman.html. [2, 4, 32, 39]

[FFH+01] Pierre Flener, Alan M. Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, and Toby Walsh. Matrix modelling. In *Proceedings of Formul'01, the CP'01 Workshop on Modelling and Problem Formulation*, 2001. [103]

[FFH+02a] Pierre Flener, Alan M. Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson, and Toby Walsh. Breaking row and column symmetries in matrix models. In *Proceedings of the 8th International Conference on Principles and Practice of*

Constraint Programming, volume 2470 of *LNCS*, 2002. [3, 30, 103]

[FFH+02b] Pierre Flener, Alan M. Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, and Toby Walsh. Matrix modelling: Exploiting common patterns in constraint programming. In *Proceedings of the International Workshop on Reformulating CSPs, held at CP'02*, 2002. [30, 105]

[FHK+02] A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Global constraints for lexicographical orderings. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 93–108, 2002. [103]

[FJHM05] Alan M. Frisch, Christopher Jefferson, Bernadette Martínez Hernández, and Ian Miguel. The rules of constraint modelling. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2005. [104]

[Gas77] J. Gaschnig. A general backtracking algorithm that eliminates most redundant tests. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, page 457, 1977. [2, 8]

[GB65] S.W. Golomb and L.D. Baumert. Backtrack programming. *Journal of the ACM*, 12(4):516–524, 1965. [2, 8]

[Ger94] C. Gervet. Conjunto: Constraint logic programming with finite set domains. In *Proceedings of the International Logic Programming Symposium*, pages 339–358, 1994. [6]

[Ger95] C. Gervet. *Set Intervals in Constraint Logic Programming: Definition and implementation of a language.* PhD thesis, Université de Franche-Comté, 1995. [2, 8, 9]

[Ger97] C. Gervet. Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints*, 1(3):191–244, 1997. [2, 6, 8, 9]

[GW99] Ian P. Gent and Toby Walsh. CSPLIB: A benchmark library for constraints. In *Principles and Practice of Constraint Programming (CP99)*, pages 480–481, 1999. Available from http://www.csplib.org/. [18, 19, 21, 23, 24, 104]

[Hen92] P. Van Hentenryck. Scheduling and packing in the constraint language cc(FD). Technical Report CS-92-43, Zweben and Fox (Eds), Morgan Kaufmann, 1992. [1]

[HF05] Bernadette Martínez Hernández and Alan M. Frisch. Towards the systematic generation of channelling constraints. In *Principles and Practice of Constraint Programming (CP)*, 2005. [104]

[HKW02] B. Hnich, Z. Kiziltan, and T. Walsh. Modelling a balanced academic curriculum problem. In *Proceedings of the 4th International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR 2002)*, pages 121–131, 2002. [23, 72]

[HPS05] B. Hnich, S. Prestwich, and E. Selensky. *Constraint-Based Approaches to the Covering Test Problem*, volume 3419 of *Lecture Notes in Computer Science*, pages 172 – 186. Springer, Mar 2005. [104]

[HSW04] B. Hnich, B. Smith, and T. Walsh. Dual modelling of permutation and injection problems. *Journal of Artificial Intelligence Research*, 21:357–391, 2004. [3, 4, 10, 18, 19, 30, 43, 45, 48, 74, 102]

[HW02] B. Hnich and T. Walsh. Models of injection problems. In *Eighth International Conference on Principles and Practice of Constraint Programming (CP)*, volume 2470 of *Lecture Notes in Computer Science*, page 781. Springer, 2002. [102, 104]

[ILO99] ILOG. *ILOG Solver 4.4, Reference Manual*, 1999. [2, 4, 6, 32, 39, 41, 49]

[Law02] Y.C. Law. Model induction: a new source of model redundancy for constraint satisfaction problems. Master's thesis, The Chinese University of Hong Kong, 2002. [104]

[Law05] Y. C. Law. Breaking value symmetries in matrix models using channeling constraints. In *Proceedings of the 20th Annual ACM Symposium on Applied Computing (SAC-2005)*, pages 375–380, 2005. [103]

[LL02] Y. C. Law and J. H. M. Lee. Model Induction: A New Source of CSP Model Redundancy. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, pages 54–60, Edmonton, Canada, 2002. [104]

[LL06] Y. C. Law and J. H. M. Lee. Symmetry breaking constraints for value symmetries in constraint satisfaction. *(to appear) Constraints*, 2006. [3, 21, 22, 24, 25, 30, 72, 103]

[Mac77] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977. [1, 2, 8, 9]

[Mil99] J. E. Miller. Langford's problem, 1999. Available from http://www.lclark.edu/ miller/langford.html. [18]

[MM88] R. Mohr and G. Masini. Good old discrete relaxation. In *Proceedings of the 8th European Conference on Artificial Intelligence*, pages 651–656, 1988. [2, 8]

[Mon74] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7(2):95–132, 1974. [2, 8]

[Moz04] Mozart. *Mozart 1.3.1, Mozart Documentation*, 2004. Available from http://www.mozart-oz.org/documentation/. [2, 4, 32, 39]

[MS98] K. Marriott and P.J. Stuckey. *Programming with Constraints*. The MIT Press, 1998. [2, 6, 108]

[Nad89] Bernard A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989. [2, 8]

[PR01] Jean-Francois Puget and Jean-Charles Régin. Solving the all interval problem, 2001. Available from http://www.csplib.org/prob/prob007/puget.pdf. [20]

[PS98] L. Proll and B. Smith. ILP and constraint programming approaches to a template design problem. *INFORMS Journal on Computing*, 10:265–275, 1998. [1]

[Rég94] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *Proceedings of the 12th National Conference* on *Artificial Intelligence (AAAI'94)*, Seattle, WA, USA, 1994. [10, 68, 70]

[SIC05] SICStus. *SICStus. SICStus Prolog, Users' Manual*, 2005. Available from http://www.sics.se/sicstus/docs/latest/html/sicstus/index.html. [2, 4, 32, 39, 41]

[SLM92] Bart Selman, Hector J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of AAAI'92*, pages 440–446, Menlo Park, California, 1992. AAAI Press. [8]

[Smi00] B. Smith. Modelling a permutation problem. In *Proceedings of ECAI'2000 Workshop on Modelling and Solving Problems with Constraints*, 2000. Also available as Research Report from http://scom.hud.ac.uk/staff/scombms/papers.html. [101]

[Smi01] B. Smith. Dual models in constraint programming. Technical report, University of Leeds, 2001. [3, 6, 16, 17, 18, 19, 21, 22, 30, 43, 45, 72, 101, 102, 104, 108]

[Smi02] B. Smith. A dual graph translation of a problem in 'Life'. In *Eighth International Conference on Principles and Practice of Constraint Programming (CP)*, volume 2470 of *LNCS*, pages 402–414, 2002. [1]

[SSW99] B. Smith, K. Stergiou, and T. Walsh. Modelling the golomb ruler problem, 1999. [1]

[Wal01] Toby Walsh. Permutation problems and channelling constraints. In *Proceedings of LPAR-2001*, volume 2250 of *LNAI*, pages 377–391. Springer, 2001. [4, 10, 43, 45, 48, 102, 104]

[ZW00] Z.Wu and B.W. Wah. An efficient global-search strategy in discrete lagrangian methods for solving hard satisfiability problems. In *Proceedings of AAAI'00*, pages 310–315, 2000. [8]