Model Induction: a New Source of Model Redundancy for Constraint Satisfaction Problems

Law Yat Chiu

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

June, 2002

Abstract

In the thesis, we formally define the concepts related to constraint satisfaction problems (CSPs). They range from CSP viewpoints to models, and model redundancy. Based on these definition, we introduce model induction, a systematic transformation of constraints in an existing model to constraints in another viewpoint. Meant to be a general CSP model operator, model induction is useful in generating redundant models, which can be further combined with the original model or other mutually redundant models. We further present two operators, namely model intersection and channeling, as two ways of combining models. Model intersection allows combining two models in the same viewpoint, while model channeling allows combining two models in different viewpoints. We identify three new forms of redundancy so as to utilize the operators to construct combined models. These combined models contain extra redundant information and hence have improved solving efficiency. We show in our benchmark results that the combined models are more robust and efficient than the original single models.

摘要

在這篇論文中，我們正式地定義了多種關於約束滿足問題(CSP)的概念，包括 CSP 觀點、模型及模型重複等。建基於這些定義上，我們介紹了模型感應。模型感應能夠有系統地將一個現有模型中的約束轉變到另一個觀點中。作為一個一般性的 CSP 模型算子，模型感應能夠製造出重複的模型。我們可以將這重複的模型與原有的或其他互相重複的模型結合。我們另外介紹了模型交集及導向，兩個用於合併模型的 CSP 算子。模型交集能夠將兩個同一觀點的模型結合，模型導向則能夠將兩個不同觀點的模型結合。此外，我們亦應用這些算子提供了三種用以發掘模型重複資訊的方式。這些方式結合同一問題上的數個模型來獲取更多的重複資訊以增加模型的解決效率。實驗結果顯示出我們結合了的模型比起個別的模型更為強健及有效率。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many problems found in artificial intelligence and computer science, such as resource allocation, scheduling, timetabling, configuration, and satisfiability problems, can be modeled as *Constraint Satisfaction Problems* (CSPs). The definition of CSP, in the sense of Mackworth [23], can be stated briefly as follows:

> *We are given a set of variables, a 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 to the variables so that all the constraints are satisfied simultaneously.*

Much CSP research effort focuses on designing general efficient (systematic or local) search algorithms for solving CSPs, and exploiting domain-specific information to solve particular applications efficiently. A recent important line of research in the community investigates how problem formulation and reformulation affect execution efficiency of constraint-solving algorithms. Freuder [11] lists first problem modeling among the seven most important future directions of constraint research.

Selecting the most appropriate formulation or model for a problem is difficult in general. In fact, no objective and general notions of the "best" formulation exist to date. Different formulations of a problem do not compete. Cheng *et al.* [8] introduce channeling constraints and present how these constraints can be used to connect mutually redundant CSP models to enhance constraint propagation in tree search. In the thesis, we formally define the concepts from CSP viewpoints to models, and model redundancy. Based on these definitions, we give another use of channeling constraints, namely to use them in generating additional model of a CSP through a process called *model induction* [22]. We give its syntactic construction rule, detailed examples, and examine its properties.

We propose the application of model induction to exploit redundant information from different models of the same problem. We introduce model intersection and channeling as two different ways to combine mutually redundant models. Model intersection [21] allows combining two models in the same viewpoint, while model channeling [8, 21] allows combining two models in different viewpoints. The latter generalizes the idea of redundant modeling [8] by combining the constraints and defining a relationship between the two viewpoints of the constituent models with the use of channeling constraints. By making use of intersection and channeling, we identify three new forms of redundancy that can enhance constraint propagation for solving CSPs. In particular, model induction generates a new model that is mutually redundant to a given one. The induced model can be combined with other models using model intersection and/or channeling to form a model that contains more redundant information. Thus, it is more plausible for the combined model to obtain enhanced constraint propagation and pruning during the search for solutions. We show in our benchmark results that the combined models are more robust and efficient.

The thesis is organized as follows. In Chapter 2, we present a brief review of the related work. These include the work related to CSP models and reformulations, and combining models. Chapter 3 provides the background to the thesis. We formally define the concepts ranging from CSP viewpoints to CSP models. We also introduce the concept of model redundancy. Besides, we present some constraint satisfaction techniques. This includes a brief overview of systematic and local search techniques for solving CSPs. In particular, we present how consistency techniques can be incorporated into backtracking search to increase CSP solving efficiency. Chapter 4 formally introduces model induction, a method for systematically generating a new model from an existing one using another viewpoint and channeling constraints. We give its syntactic construction rule, detailed examples, and examine its properties. We present three new forms of model redundancy based on model induction in Chapter 5. We introduce model intersection and channeling as two ways for combining models. Our proposal utilizes model intersection, channeling, and induction to combine models to form mutually redundant ones that contain more redundant information for enhancing constraint propagation during search. We also present experimental results on combining mutually redundant models using our proposed scheme. Finally, we conclude the thesis in Chapter 6 by giving our contributions and possible directions of future research.

Chapter 2

Related Work

In this chapter, we present the research conducted that is related to our work on combining and transforming models. The related work can be classified as two types: CSP models and reformulations, and combining models. The following three sections correspond to CSP models and reformulations, and the next four sections correspond to combining models.

2.1 Equivalence of CSPs

Rossi *et al.* [31] propose a new definition of equivalence of CSPs, based on the concept of mutual reducibility. They believe that it is reasonable to consider two CSPs equivalent if it is possible to obtain the solution of one CSP from that of another, and vice versa. Based on their definition of equivalence of CSPs, they formally address the issue of the equivalence of binary and non-binary CSPs. Their definition makes a fundamental contribution in that it is an appropriate tool to identify redundant information in CSPs.

2.2 Dual Viewpoint

Geelen [13] introduces two improved problem-independent value and variable ordering heuristics for solving CSPs. He also introduces a "dual-viewpoint" approach for a special but broad class of CSPs, namely Permutation CSPs. This approach allows suitable extensions to many heuristics including those introduced in his paper. We can consider a Permutation CSP from two different perspectives. In one perspective, we solve the problem by finding a value for every variable, while in another perspective, we solve the problem by finding a variable for each value. Geelen improves the most-constrained-first variable ordering heuristic by calculating and incorporating the constrainedness information from both viewpoints. Hence, the number of backtracks can be reduced.

2.3 CSP Reformulation

Weigel and Bliek [40] introduces an algorithm to transform a CSP into its boolean form which is then used to find its reformulations. Reformulations differ with each other only in redundant constraints, and one can allow pruning in some situations which is not possible in other. They identify a new class of tractable CSPs and sufficient conditions for deciding solvability and unsolvability of a CSP in linear time. They also introduce the notion of fault tolerance of solutions, and show how the boolean form can be used to find them.

2.4 Multiple Modeling

Jourdan [19, 20] works on multiple modeling, in which models representing different but redundant views of the same problem are synchronized using the communication mechanisms of constraint logic programming and concurrent constraint languages. Besides redundant models, Jourdan also explores the notions of cooperating and hierarchical models.

2.5 Redundant Modeling

Cheng *et al.* [5, 6, 7, 8] formally introduces redundant modeling. Two models of the same problem are combined together using channeling constraints. The combined model contains the mutually redundant models as sub-models. Channeling constraints allow the sub-models to cooperate during constraint solving by allowing constraint propagation to take place among the sub-models. They show increased constraint propagation and efficiency by using this approach to a real life nurse rostering problem. Besides, redundant modeling can be incorporated with Jourdan's new value ordering heuristic [20] to open up new horizon for the definition of new value ordering heuristics.

2.6 Minimal Combined Model

Smith [36, 37] introduces the idea of minimal combined models for Permutation CSPs. It is similar to redundant modeling but the constraints in the second model are dropped. She shows that for the Langford's problem, the amount of constraint propagation of the minimal combined model is equal to that of redundant modeling. Since the constraints of the second model are dropped, the time required for constraint propagation is shorter than that in redundant modeling. Therefore, the execution time for a minimal combined model would be shorter than that for using redundant modeling. However, it is not clear whether the same result can be transferred to Permutation CSPs in general.

2.7 Permutation CSPs and Channeling Constraints

Walsh [39] conducts an extensive theoretical and empirical study on using different models and combined models using channeling constraints. He compares models by defining a measure of constraint tightness by the level of consistency being enforced for the constraints in a model. His results aid human CSP modelers to choose a viewpoint for Permutation CSPs. He also illustrates a general methodology for comparing different CSP models. Smith and Walsh's works concentrate on the effect of different levels of constraint propagation on the constraints in a model to ensure a permutation in Permutation CSPs.

Chapter 3

Background

This chapter provides background to the thesis. We provide the basic definitions relating from viewpoints to CSP models. Furthermore, a description of constraint satisfaction techniques is presented. This includes a brief overview of systematic and local search techniques for solving CSPs. In particular, we present how consistency techniques can be incorporated into backtracking search to increase CSP solving efficiency.

3.1 From Viewpoints to CSP Models

There are usually more than one way of formulating a problem P into a CSP. Central to the formulation process is to determine the variables and the domains (associated sets of possible values) of the variables. Different choices of variables and domains are results of viewing the problem P from different angles/perspectives. We define a *viewpoint*[1] to be a pair (X, D_X), where $X = \{x_1, \ldots, x_n\}$ is a set of variables, and D_X is a set containing, for every $x \in X$, an associated domain $D_X(x)$ giving the set of possible values for x.

A viewpoint $V = (X, D_X)$ defines the possible assignments for variables in X. An *assignment* in V (or in $U \subseteq X$) is a pair $\langle x, a \rangle$, which means that

[1] Geelan [13] used the notion of viewpoint loosely without actually defining it.

variable $x \in X$ (or U) is assigned the value $a \in D_X(x)$. A *compound assignment* in V (or in $U \subseteq X$) is a set of assignments $\{\langle x_{i_1}, a_1\rangle, \ldots, \langle x_{i_k}, a_k\rangle\}$, where $\{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ (or U) and $a_j \in D_X(x_{i_j})$ for each $j \in \{1, \ldots, k\}$. *Note* the requirement that *no* variables may be assigned more than one value in a compound assignment. Given a set of assignments θ, we use the predicate $cmpd(\theta, V)$ to ensure that θ is a compound assignment in V. A *complete assignment* in V is a compound assignment $\{\langle x_1, a_1\rangle, \ldots, \langle x_n, a_k\rangle\}$ for all variables in X.

When formulating a problem P into a CSP, the choice of viewpoints is not arbitrary. Suppose $sol(P)$ is the set of all solutions of P (in whatever notations and formalism). We say that viewpoint V is *proper* for P if and only if we can find a subset S of the set of all possible complete assignments in V so that there is a one-one mapping between S and $sol(P)$. In other words, each solution of P must correspond to a distinct complete assignment in V. We note also that according to our definition, any viewpoint is proper with respect to a problem that has no solutions. The definition of equivalence of CSPs by Rossi *et al.* [31] produce a similar effect on problems with no solutions. According to their definition, any CSPs with no solutions are equivalent.

A *constraint* can be considered a predicate that maps to *true* or $false$. The *signature* $sig(c) \subseteq X$, which is the set of variables involved in c, defines the scope of c. We abuse terminology by saying that the compound assignment $\{\langle x_{i_1}, a_1\rangle, \ldots, \langle x_{i_k}, a_k\rangle\}$ also has a signature: $sig(\{\langle x_{i_1}, a_1\rangle, \ldots, \langle x_{i_k}, a_k\rangle\}) = \{x_{i_1}, \ldots, x_{i_k}\}$. Given a compound assignment θ such that $sig(c) \subseteq sig(\theta)$, the *application* of θ to c, $c\theta$, is obtained by replacing all variables in c by the corresponding values in θ. If $c\theta$ is *true*, we say θ *satisfies* c, and θ *violates* c otherwise. In addition, the negation $\neg c$ of a constraint c is defined by the fact that $(\neg c)\theta = \neg(c\theta)$ for all compound assignments θ in $X \supseteq sig(c)$. We overload the $\neg$ operator so that it operates on both constraints and boolean expressions. A constraint c is said to be *unary* if and only if $|sig(c)| = 1$, and *binary* if and only if it is unary or $|sig(c)| = 2$.

A *CSP model* M (or simply *model* hereafter) of a problem P is a pair (V, C), where V is a proper viewpoint of P and C is a set of constraints in V for P. Note that, in our definition, we allow two constraints to be on the same set of variables: $c_i, c_j \in C$ and $sig(c_i) = sig(c_j)$. A CSP model is said to be *binary* if and only if all its constraints are binary. A *solution* of $M = (V, C)$ is a complete assignment θ in V so that $c\theta = true$ for every $c \in C$. Since M is a model of P, the constraints C must be defined in such a way that there is a one-one correspondence between $sol(M)$ and $sol(P)$. Thus, the viewpoint V essentially dictates how the constraints of P are formulated (*modulo* solution equivalence). A model M is *satisfiable* or *soluble* if $sol(M) \neq \emptyset$, and *unsatisfiable* or *insoluble* otherwise.

Suppose M_1 and M_2 are two different models of the same problem P. By definition, there exists a one-one mapping between $sol(M_1)$ and $sol(M_2)$. We say that M_1 and M_2 are *mutually redundant*. As we shall see, it is possible for mutually redundant models M_1 and M_2 to share the same viewpoint. In that special case, it is easy to verify that $sol(M_1) = sol(M_2)$.

3.2 Constraint Satisfaction Techniques

Since CSPs are NP-complete [9] in general, any algorithms for solving CSPs are likely to require exponential time in problem size in the worst case. There are two general classes of algorithms for solving CSPs. The first class of algorithms is systematic search, which enumerates through the possible assignments of the variables. This class of algorithms is guaranteed to find a solution, if there is any, or prove no solutions exist. Therefore, it is sound and complete. Another class of algorithms is local search. A local search algorithm typically starts with a complete assignment of the CSP. It then incrementally alters the assignments until a solution is found. These algorithms use the hill-climbing or repair heuristics to move towards the solutions. Local search algorithms can be trapped in local optima, in which no altering of assignments can be made to move towards the solutions. Therefore, local search algorithms must have some mechanisms to escape from such situations. A random restart or modification of the landscape of the search surface can be useful. Local search algorithms are usually incomplete. In other words, they may fail to return a solution even if one exists in the CSP. Furthermore, they cannot prove that a CSP has no solutions in general. The work reported in this thesis concerns systematic search algorithms.

The most common algorithm for performing systematic search is backtracking [15, 10, 3, 12, 28]. To improve the efficiency of backtracking search, consistency and propagation techniques are used to remove inconsistent domain values, aiming at detecting failures earlier in the search. In the following subsections, we describe briefly backtracking search, consistency and propagation techniques, and incorporation of backtracking search and constraint propagation for solving CSPs.

3.2.1 Backtracking Search

Backtracking search [15, 10, 3, 12, 28] is a common algorithm for performing systematic search to solve a CSP. It starts with an empty compound assignment. It incrementally extends the compound assignment by choosing an unselected variable and making an assignment to this variable from the variable's domain. If the new compound assignment violates some of the constraints of the CSP, the search backtracks and tries another assignment of the variable. If there are no more assignments to try for this variable, the search further backtracks to the previously chosen variable and tries another assignment of that variable. This process is repeated until either a solution is found, or there are no more variables to backtrack to. In the latter case, backtracking search

$backtracking\_search(((X, D), C), \theta)$
  if $sig(\theta) = X \vee C = \emptyset$ **then**
    return $true$
  choose $x \in X \setminus sig(\theta)$
  for each $a \in D(x)$
    $\theta := \theta \cup \{\langle x, a\rangle\}$
    $C' := \{c | c \in C \wedge sig(c) \subseteq sig(\theta)\}$
    if $\forall c \in C' \cdot [c\theta] = true$ **then**
      if $backtracking\_search(((X, D), C \setminus C'), \theta) = true$ **then**
        return $true$
    $\theta := \theta \setminus \{\langle x, a\rangle\}$
  return $false$

Figure 3.1: Backtracking Search Algorithm

proves that no solutions exist for this CSP. Figure 3.1 shows the algorithm for backtracking search [24]. The call $backtracking\_search(M, \theta)$ with $\theta = \emptyset$ would start searching a solution for the model M. If a solution is found, the search returns $true$, and $false$ otherwise.

3.2.2 Consistency Techniques and Constraint Propagation

Consistency techniques are algorithms that remove values from the domains of variables without removing any solution to a CSP. The idea behind these techniques is based on the observation that if a value in a variable domain cannot satisfy a constraint in a model, then the value must not be in any solution of the model. There are several kinds of consistency techniques, which behave differently upon different kinds of constraints. Removal of a variable's domain value by maintaining consistency of one constraint may affect the consistency of another in a CSP, and make other values be removed from other variables. Constraint propagation is the spreading of pruning information from one constraint to another. In the following, we briefly describe node consistency, arc consistency, and bounds consistency, consistency algorithms

node_consistent$(((X, D), C))$
  for each $c \in C$ with $|sig(c)| = 1$
    let $sig(c) = \{x\}$
    $D(x) := \{a | a \in D(x) \wedge c\{\langle x, a\rangle\} = true\}$
  return D

Figure 3.2: Node Consistency Algorithm

on constraints, and constraint propagation which maintains these consistencies in a model level.

Node Consistency

A constraint c is *node consistent* [24] with respect to viewpoint $V = (X, D)$ if either $|sig(c)| \neq 1$ or if $sig(c) = \{x\}$, then for each $j \in D(x)$, $c\{\langle x, j\rangle\}$ is true. A CSP (V, C) is *node consistent* if each constraint $c \in C$ is node consistent with respect to V.

Example 3.1 Suppose we have $D(x) = \{0, 1, 2, 3, 4\}$. Then the constraint $x < 3$ is *not* node consistent, or *node inconsistent*, because the assignments $\langle x, 3\rangle$ and $\langle x, 4\rangle$ do not satisfy the constraint. If we remove values 3 and 4 from $D(x)$, i.e., $D(x) = \{0, 1, 2\}$, then the constraint becomes node consistent. □

The algorithm that transforms a CSP into its node consistent counterpart is straightforward. For each unary constraint with signature $\{x\}$, if a value in the domain of x violates the constraint, we remove the value from the variable domain. Figure 3.2 shows the algorithm that makes a CSP model node consistent [24]. The algorithm returns the new domains of the variables.

Node consistency can be achieved as a preprocessing step before starting the search. Once it is achieved, all unary constraints are always satisfied because the inconsistent values are already removed. Therefore, these unary constraints can be discarded/ignored.

$ac\_revise((X, D), c)$
  if $|sig(c)| = 2$ **then**
    let $sig(c) = \{x, y\}$
    $D(x) := \{a | a \in D(x) \wedge \exists b \in D(y) \cdot [c\{\langle x, a\rangle, \langle y, b\rangle\}]\}$
    $D(y) := \{b | b \in D(y) \wedge \exists a \in D(x) \cdot [c\{\langle x, a\rangle, \langle y, b\rangle\}]\}$
  return D

Figure 3.3: Algorithm Making a Constraint Arc Consistent

Arc Consistency

A constraint c is *arc consistent* [24] with respect to viewpoint $V = (X, D)$ if either $|sig(c)| \neq 2$ or if $sig(c) = \{x, y\}$, then for each $j \in D(x)$ there exists $k \in D(y)$ such that $c\{\langle x, j\rangle, \langle y, k\rangle\}$ is true, and for each $k \in D(y)$ there exists $j \in D(x)$ such that $c\{\langle x, j\rangle, \langle y, k\rangle\}$ is true. A CSP (V, C) is *arc consistent* if each constraint $c \in C$ is arc consistent with respect to V.

Example 3.2 Suppose we have $D(x) = D(y) = \{0, 1, 2, 3, 4\}$. The constraint $x < y$ is *not* arc consistent, or *arc inconsistent*, because for $4 \in D(x)$ there does not exist a value in $D(y)$ such that the constraint is satisfied. Similarly, for $0 \in D(y)$ there does not exist a value in $D(x)$ such that the constraint is satisfied. If we remove 4 and 0 from $D(x)$ and $D(y)$ respectively, i.e., $D(x) = \{0, 1, 2, 3\}$ and $D(y) = \{1, 2, 3, 4\}$, then the constraint becomes arc consistent. □

To make a constraint c arc consistent, we can simply delete all values from the domain of the variables in the constraint that cannot satisfy c. Figure 3.3 shows the algorithm that removes values from the variable domains to make a constraint arc consistent [24].

To make a CSP arc consistent, it is not enough to revise each constraint in the CSP only once. When the revise algorithm removes a value from the domain of a variable, other previously revised constraints may become arc inconsistent again. The spreading of pruning information from one constraint to another is called *constraint propagation*. The AC-1 algorithm [24], shown in

$ac\_1((V,C))$
  repeat
    $W := D$
    for each $c \in C$
      $D := ac\_revise(V, c)$
  until $W = D$
  return D

Figure 3.4: Arc Consistency Algorithm AC-1

$ac\_3((V,C))$
  $Q := C$
  while $Q \neq \emptyset$
    delete a constraint $c \in Q$
    $W := D$
    $D := ac\_revise(V, c)$
    if $W \neq D$ **then**
      $Q := Q \cup \{c' | c' \in C \wedge sig(c) \cap sig(c') \neq \emptyset\}$
  return D

Figure 3.5: Arc Consistency Algorithm AC-3

Figure 3.4, performs constraint propagation and makes a CSP arc consistent by revising all the constraints again and again until there are no changes to the variable domains.

The AC-1 algorithm is quite inefficient because domain reduction of one variable makes all constraints in the model be revised again. It is clear that this is not necessary since when a variable has its domain reduced, only constraints with this variable are affected. The AC-3 algorithm [23] in Figure 3.5 is an improved version of AC-1. It only re-revises those constraints that can possibly be affected rather than re-revising all of them.

There are more sophisticated arc consistency algorithms such as AC-4 [26], AC-5 [29], AC-6 [1], and AC-7 [2] but their basic ideas are the same: to remove values from domains of variables that cannot contribute solutions to the CSP.

Bounds Consistency

Node consistency and arc consistency handles unary and binary constraints in a CSP only. When we have constraints in a model with more than two variables, these consistency algorithms cannot reduce the variable domains. *Generalized arc consistency* (GAC) [27] is the generalization of arc consistency to non-binary constraints. A constraint c is *generalized arc consistent* [27] with respect to viewpoint $V = (X, D)$ if for each $x \in sig(c)$ and $j \in D(x)$, there are assignments $\langle x_i, d_i \rangle$ to the variables $x_i \in sig(c) \setminus \{x\}$ such that $c(\{\langle x, j \rangle\} \cup \{\langle x_i, d_i \rangle | x_i \in sig(c) \setminus \{x\} \wedge d_i \in D(x_i)\})$ is true. A CSP (V, C) is *generalized arc consistent* if each constraint $c \in C$ is generalized arc consistent with respect to V.

There are algorithms that can maintain generalized arc consistency for non-binary constraints. However, the complexities of these algorithms are usually high for general constraints that they are not practical to be incorporated in a CSP solver. Therefore, algorithms that maintain weaker levels of consistency arise. They may reduce fewer values from the variable domains than those generalized arc consistency algorithms, but they have a relatively low complexity to execute. Bounds consistency [24] is a weaker level of consistency but it works for integer variable domains and non-binary arithmetic constraints at a low cost.

Suppose we have a viewpoint $V = (X, D)$. We define $min_D(x)$ and $max_D(x)$ to be the minimum and maximum element in $D(x)$ respectively. An *arithmetic constraint* c is *bounds consistent* [24] with respect to V if for each $x \in sig(c)$, the followings are true:

- There are assignments of *real* numbers $\langle x_i, d_i \rangle$ to the variables $x_i \in sig(c) \setminus \{x\}$ such that $c(\{\langle x, min_D(x) \rangle\} \cup \{\langle x_i, d_i \rangle | x_i \in sig(c) \setminus \{x\} \wedge min_D(x_i) \leq d_i \leq max_D(x_i)\})$ is true.

- There are assignments of *real* numbers $\langle x_i, d_i' \rangle$ to the variables $x_i \in$

$sig(c) \setminus \{x\}$ such that $c(\{\langle x, max_D(x)\rangle\} \cup \{\langle x_i, d'_i\rangle | x_i \in sig(c) \setminus \{x\} \wedge min_D(x_i) \leq d'_i \leq max_D(x_i)\})$ is true.

Note that we relax the definition of assignments here by allowing the value of an assignment not in the domain of the variable. A CSP (V, C) is *bounds consistent* if each constraint $c \in C$ is bounds consistent with respect to V.

Example 3.3 Suppose we have $D(x) = D(y) = D(z) = \{0, 1, 2, 3\}$. The arithmetic constraint $x - 1 = y + z$ is *not* bounds consistent, or *bounds inconsistent*. Consider the value $min_D(x) = 0 \in D(x)$. We cannot find values within $[0 \ldots 3]$ for y and z such that $y + z = 0 - 1 = -1$. If we remove 0 from $D(x)$, the domain becomes $D(x) = \{1, 2, 3\}$ and $min_D(x) = 1$. The constraint now becomes bounds consistent. □

Bounds consistency of a constraint is maintained using propagation rules. Consider the simple constraint $x + y = z$. We can easily derive the following inequalities:

- $x \geq min_D(z) - max_D(y)$, $x \leq max_D(z) - min_D(y)$
- $y \geq min_D(z) - max_D(x)$, $y \leq max_D(z) - min_D(x)$
- $z \geq min_D(x) + min_D(y)$, $z \leq max_D(x) + max_D(y)$

These inequalities produce propagation rules for maintaining bounds consistency of the constraint $x + y = z$. Figure 3.6 shows the algorithm that removes values from the variable domains to make this constraint bounds consistent [24]. The propagation rules for constraints other than $x+y = z$ can be derived similarly. Hence, bounds consistency provides an efficient way to remove values from variable domains even the constraint has many variables. However, when a constraint has only two variables, maintaining arc consistency may potentially remove more values than maintaining bounds consistency. For example, suppose $D(x) = D(y) = \{0, \ldots, 10\}$. Maintaining arc consistency on the

$bc\_revise\_add(D)$
$D(x) := \{a | a \in D(x) \wedge min_D(y) + min_D(z) \leq a \leq max_D(y) + max_D(z)\}$
$D(y) := \{a | a \in D(y) \wedge min_D(x) - max_D(z) \leq a \leq max_D(x) + min_D(y)\}$
$D(z) := \{a | a \in D(z) \wedge min_D(x) - max_D(y) \leq a \leq max_D(x) + min_D(y)\}$
return D

Figure 3.6: Bounds Consistency Algorithm for the Constraint $x + y = z$

constraint $2x = 3y+1$ makes $D(x) = \{2, 5, 8\}$ and $D(y) = \{1, 3, 5\}$, while maintaining bounds consistency on the same constraint makes $D(x) = \{2, \ldots, 8\}$ and $D(y) = \{1, \ldots, 5\}$[33]. Therefore, bounds consistency is a weaker level of consistency than arc consistency when handling binary constraints.

3.2.3 Incorporating Consistency Techniques into Backtracking Search

Although backtracking search is more efficient than pure "generate and test" for solving CSPs, its performance is still not good enough to tackle CSPs. To improve efficiency, consistency techniques are incorporated into backtracking search [17, 32, 16, 2]. The idea is that in backtracking search, after choosing a variable and making assignment to a value for this variable from the variable's domain, constraint propagation is performed to reduce the domains of the unassigned variables. If the domain of a variables becomes empty after propagation, the current assignment must not lead to a solution and backtracking occurs. The removal of domain values is undone upon backtracking, and the search makes another assignment for the variable. This process is continued until either a solution is found, or there are no more variables to backtrack to.

We give an example of performing forward checking [17] during search using the n-queens problem. Forward checking maintains arc consistency of the constraints between the currently assigned variable and the unassigned variables in the search node. The n-queens problem is to place n queens on a chessboard such that no two queens can attack each other. That means there can be no two queens on each row, each column, and each diagonal. We give a textbook model $M = ((X, D_X), C_X)$ of the 4-queens problem. We use four variables $X = \{x_1, x_2, x_3, x_4\}$ and their associated domain function D_X. Each x_i denotes the column position of the queen on row i and $D_X(x_i) = \{1, 2, 3, 4\}$ for $i \in \{1, 2, 3, 4\}$. The constraints C_X enforce that no two queens can be on the same:

- column: $x_i \neq x_j$ for all $1 \leq i < j \leq 4$, and
- diagonal: $|x_i - x_j| \neq j - i$ for all $1 \leq i < j \leq 4$.

Figure 3.7 shows the search tree of the 4-queens problem performing forward checking. When the search starts, the CSP is already arc consistent. Therefore, no values are removed. Suppose we are trying the variables from x_1 to x_4 and the values 1 to 4 in order. By trying the assignment $\langle x_1, 1\rangle$, constraint propagation removes values that are on the same row and diagonal of the corresponding queen. Next, the search makes the assignment $\langle x_2, 3\rangle$. Constraint propagation removes all the values from the domain of x_3; hence backtracking occurs and another value in $D_X(x_2)$ is tried. Then the search makes the assignment $\langle x_2, 4\rangle$. After propagation, only one value in $D_X(x_3)$ is left. Trying the assignment $\langle x_3, 2\rangle$ makes the domain of x_4 empty. Therefore, the search backtracks again. This time, there are no values in the domains of x_3 and x_2 to be tried anymore, we have to try another value of x_1. After trying $\langle x_1, 2\rangle$ and propagation, there is only one value left in $D_X(x_2)$. By assigning x_2 to 4, x_3 must be assigned 1 and then x_4 be assigned 3. In this state, the search obtains a solution $\{\langle x_1, 2\rangle, \langle x_2, 4\rangle, \langle x_3, 1\rangle, \langle x_4, 3\rangle\}$ of the 4-queens problem with two backtracks. If we want to find another solution of the problem, the search would also backtrack to start finding another solution.

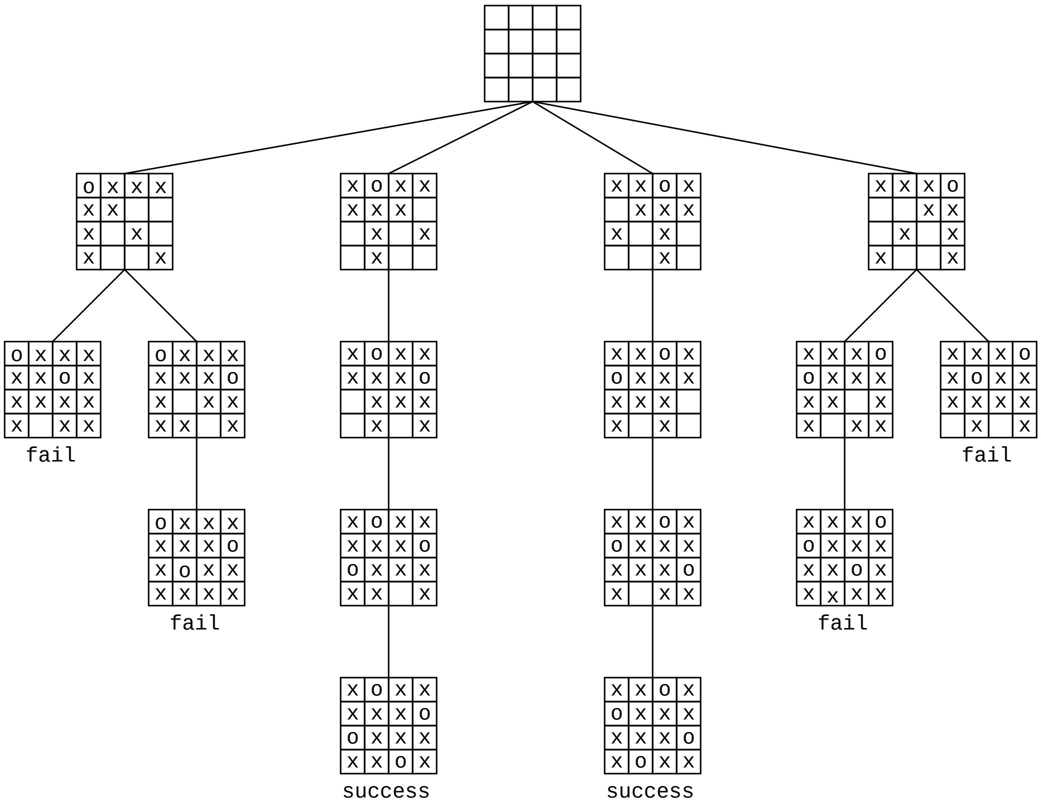


Figure 3.7: Forward Checking on the n-queens Problem during Backtracking Search

Chapter 4

Model Induction

In this chapter, we introduce *model induction* [22]: a method for systematically generating a new model from an existing model, using another viewpoint and channeling constraints. The resulting model is called an *induced model*. The core of model induction is a meaning-preserving transformation for constraints, both implicit and explicit, from one model to constraints in another viewpoint. In the following, we describe channeling constraints, construction of induced models with detailed examples, and properties of model induction.

4.1 Channeling Constraints

Given two models $M_1 = ((X, D_X), C_X)$ and $M_2 = ((Y, D_Y), C_Y)$. Cheng *et al.* [8] define a *channeling constraint* c to be a constraint, where $sig(c) \not\subseteq X$, $sig(c) \not\subseteq Y$, and $sig(c) \subseteq X \cup Y$. Thus, c relates M_1 and M_2 by limiting the combination of values that their variables can take. Cheng *et al.* show how a collection of channeling constraints can be used to connect two mutually redundant models of the same problem to form a combined model, which exhibits increased constraint propagation and thus improved efficiency.

We note in the definition that the constraints in the two models are immaterial. Channeling constraints relate actually the viewpoints of the models. In other words, channeling constraints set forth a relationship between the possible assignments of the two viewpoints. Not all arbitrary sets of channeling constraints can be used in model induction. Given viewpoints $V_1 = (X, D_X)$ and $V_2 = (Y, D_Y)$, suppose we want to construct an induced model from a model in viewpoint V_1 to another viewpoint V_2. A necessary condition is that the set of channeling constraints between V_1 and V_2 must collectively define a *total* and *injective* function f from the possible assignments in V_1 to those in V_2:

$$f : \{\langle x, a\rangle \mid x \in X \wedge a \in D_X(x)\} \rightarrow \{\langle y, b\rangle \mid y \in Y \wedge b \in D_Y(y)\}.$$

In other words, f maps every assignment in V_1 to a unique assignment in V_2.

4.2 Induced Models

Description of model induction assumes constraints to be represented extensionally. We define $rel(c)$ to be the *relation* of a constraint c, and $rel(c)$ is stored explicitly as a set of incompatible assignments in $sig(c)$. Suppose $sig(c) = \{x_{i_1}, \ldots, x_{i_k}\}$, an *incompatible assignment* $\{\langle x_{i_1}, a_1\rangle, \ldots, \langle x_{i_k}, a_k\rangle\}$ for c is a compound assignment in $sig(c)$ that violates c. Otherwise, it is a *compatible assignment*. The incompatible assignment has the logical meaning $\neg((x_{i_1} = a_1) \wedge \ldots \wedge (x_{i_k} = a_k))$. Hence, a constraint check amounts to a set membership check against $rel(c)$: $c\theta \Leftrightarrow \pi_{sig(c)}(\theta) \notin rel(c)$ for all valid θ.

Given a model $M = ((X, D_X), C_X)$, a viewpoint (Y, D_Y), and a set of channeling constraints defining a total and injective function f from the possible assignments in (X, D_X) to those in (Y, D_Y). We note that a CSP M contains two types of constraints: the explicit constraints as stated in C_X and the implicit constraints on variable assignments. The latter type of constraints can be further broken down into the restriction that (1) each variable must be assigned a value from its associated domain and (2) each variable cannot be assigned more than one value from its domain. The idea of model induction is to transform the constraints in model M, both implicit and explicit, using f to constraints C_Y in viewpoint (Y, D_Y), yielding the induced model $i(f, M) = ((Y, D_Y), C_Y)$.

- **Stated Constraints.** The first type of constraints to transform is the constraints stated in C_X. Recall that a constraint c consists of a signature and a relation, which is simply a set of incompatible assignments for c. We apply f on the assignments in each incompatible assignments of all constraints in C_X, and collect the transformed incompatible assignments in a set S_Y:

$$\begin{aligned} S_Y &= \{\theta | \theta = \{f(\langle x_{i_1}, a_1\rangle), \ldots, f(\langle x_{i_k}, a_k\rangle)\} \wedge c \in C \wedge \\ &\qquad \{\langle x_{i_1}, a_1\rangle, \ldots, \langle x_{i_k}, a_k\rangle\} \in rel(c) \wedge cmpd(\theta, (Y, D_Y))\}. \end{aligned}$$

  It is indeed possible for θ not being a compound assignment with, say, $f(\langle x_{i_u}, a_u\rangle)$ and $f(\langle x_{i_v}, a_v\rangle)$ being $\langle y, b_u\rangle$ and $\langle y, b_v\rangle$, where $y \in Y$ and $b_u \neq b_v$. Since we are transforming incompatible assignments from (X, D_X), the information conveyed in θ, including the restriction that the variable y cannot be assigned values b_u and b_v simultaneously, is correct. In fact, this information is already satisfied implicitly in viewpoint (Y, D_Y) so that we can ignore/discard θ.

  Example 4.1 Suppose we have a constraint $c \in C_X$ with $\{\langle x_1, 1\rangle, \langle x_2, 2\rangle\} \in rel(c)$. If $f(\langle x_1, 1\rangle) = \langle y_2, 1\rangle$ and $f(\langle x_2, 2\rangle) = \langle y_1, 1\rangle$, then we have $\{f(\langle x_1, 1\rangle), f(\langle x_2, 2\rangle)\} = \{\langle y_1, 1\rangle, \langle y_2, 1\rangle\} \in S_Y$. If $\{\langle x_1, 1\rangle, \langle x_2, 1\rangle\} \in rel(c)$ and $f(\langle x_2, 1\rangle) = \langle y_2, 2\rangle$, then the set $\{f(\langle x_1, 1\rangle), f(\langle x_2, 1\rangle)\} = \{\langle y_2, 1\rangle, \langle y_2, 2\rangle\}$ is not a compound assignment and is not in S_Y. □

- **No-Double-Assignment Constraints.** Implicit in a CSP formulation, each variable should be assigned exactly one value. Part of this restriction can be translated to the requirement that no variables can be assigned two values from its domain at the same time. This corresponds to a set of (invalid) incompatible assignments of the form $\{\langle x, a\rangle, \langle x, b\rangle\}$ for all $x \in X$, $a, b \in D_X(x)$, and $a \neq b$, which is satisfied implicitly and not represented in M. Their transformed counterparts, however, are needed in (Y, D_Y). We apply f on all these assignment sets, and collect the transformed incompatible assignments in a set N_Y:

$$\begin{aligned} N_Y \;\; &= \textstyle\bigcup_{x\in X} \;\; \{\{f(\langle x, a\rangle), f(\langle x, b\rangle)\} \mid a, b \in D_X(x) \wedge a \neq b \wedge \\ &\qquad\qquad cmpd(\{f(\langle x, a\rangle), f(\langle x, b\rangle)\}, (Y, D_Y))\} \end{aligned}$$

Example 4.2 Suppose $D_X(x_1) = \{2, 3\}$, with $f(\langle x_1, 2\rangle) = \langle y_1, 2\rangle$ and $f(\langle x_1, 3\rangle) = \langle y_2, 1\rangle$. Then, for variable x_1, $\{f(\langle x_1, 2\rangle), f(\langle x_1, 3\rangle)\} = \{\langle y_1, 2\rangle, \langle y_2, 1\rangle\} \in N_Y$. □

- **At-Least-One-Assignment Constraints.** The other part of the implicit variable constraint in M can be translated to the requirement that each variable must be assigned at least a value from its domain. This corresponds to the constraints $\bigvee_{b\in D_X(x)} x = b$ for all $x \in X$, which are satisfied implicitly and not represented in M. The other problem is that this unary constraint does not have any incompatible assignments. For each variable $x \in X$, we first apply f to every possible assignment of x. Suppose $D_X(x) = \{b_1, \ldots, b_r\}, f(\langle x, b_1\rangle) = \langle y_{k_1}, v_1\rangle, \ldots, f(\langle x, b_r\rangle) = \langle y_{k_r}, v_r\rangle$. These assignments form the compatible assignments of a constraint in $\{y_{k_1}, \ldots, y_{k_r}\}$. Using the closed world assumption [30], we compute the incompatible assignments by collecting all compound assignments θ with signature $\{y_{k_1}, \ldots, y_{k_r}\}$ such that every individual assignment in θ is not equal to $\langle y_{k_j}, v_j\rangle$ for all $j \in \{1, \ldots, r\}$:

$$\begin{aligned} A_Y \;= \bigcup_{x \in X} \; & \{\theta \mid D_X(x) = \{b_1, \ldots, b_r\} \wedge \\ & \forall j \in \{1, \ldots, r\} \cdot [f(\langle x, b_j\rangle) = \langle y_{k_j}, v_j\rangle] \wedge \\ & \theta = \{\langle y_{w_1}, a_1\rangle, \ldots, \langle y_{w_s}, a_s\rangle\} \wedge cmpd(\theta, (Y, D_Y)) \wedge \\ & sig(\theta) = \{y_{k_1}, \ldots, y_{k_r}\} \wedge \\ & \forall i \in \{1, \ldots, s\}, \forall j \in \{1, \ldots, r\} \cdot [\langle y_{w_i}, a_i\rangle \neq \langle y_{k_j}, v_j\rangle]\} \end{aligned}$$

Example 4.3 Suppose we have $D_X(x_1) = D_Y(y_1) = D_Y(y_2) = \{1, 2, 3\}$ with $f(\langle x_1, 1\rangle) = \langle y_1, 1\rangle$, $f(\langle x_1, 2\rangle) = \langle y_1, 2\rangle$, and $f(\langle x_1, 3\rangle) = \langle y_2, 1\rangle$. Then, for variable x_1, applying f to the assignments $\langle x_1, 1\rangle, \langle x_1, 2\rangle$, and $\langle x_1, 3\rangle$ suggests that the incompatible assignments are among y_1 and y_2. All compound assignments $\{\langle y_1, a\rangle, \langle y_2, b\rangle\}$ where $a \neq 1, a \neq 2$, and $b \neq 1$ are the incompatible assignments in A_Y. Hence, we have $\{\langle y_1, 3\rangle, \langle y_2, 2\rangle\}$ and $\{\langle y_1, 3\rangle, \langle y_2, 3\rangle\}$ for inclusion in A_Y. □

We note that the incompatible assignments in a constraint $c \in C_X$ may be transformed to contribute to the incompatible assignments of more than one constraint in (Y, D_Y). Thus $S_Y \cup N_Y \cup A_Y$ consists of all the induced incompatible assignments with different signatures in (Y, D_Y). The next step is to extract incompatible assignments with the same signature from $S_Y \cup N_Y \cup A_Y$ and group them into a constraint in (Y, D_Y). Thus,

$$C_Y = \{c \mid sig(c) \subseteq Y \wedge rel(c) = \sigma_{sig(c)}(S_Y \cup N_Y \cup A_Y) \neq \emptyset\},$$

where $\sigma_U(\Theta) = \{\theta \mid \theta \in \Theta \wedge sig(\theta) = U\}$.

We illustrate the construction of two induced models using the simple 4-queens problem introduced in Chapter 3. The 4-queens problem is to place four queens on a 4×4 chessboard in such a way that no two queens can attack each other.

Recall our textbook model $M = ((X, D_X), C_X)$ of the 4-queens problem. We use four variables $X = \{x_1, x_2, x_3, x_4\}$ and their associated domain function D_X. Each x_i denotes the column position of the queen on row i and $D_X(x_i) = \{1, 2, 3, 4\}$ for $i \in \{1, 2, 3, 4\}$. The constraints C_X enforce that no two queens can be on the same:

- column: $x_i \neq x_j$ for all $1 \leq i < j \leq 4$, and
- diagonal: $|x_i - x_j| \neq j - i$ for all $1 \leq i < j \leq 4$.

Example 4.4 We consider a 0-1 viewpoint (Z, D_Z) with sixteen variables $Z = \{z_{ij} \mid i, j \in \{1, 2, 3, 4\}\}$ and associated domain function D_Z. The assignment $\langle z_{ij}, 1\rangle$ denotes the fact that square position (i, j) (row i and column j) contains a queen; and $\langle z_{ij}, 0\rangle$ denotes otherwise. Therefore, $D_Z(z_{ij}) = \{0, 1\}$ for all $i, j \in \{1, 2, 3, 4\}$. The set of channeling constraints $x_i = j \Leftrightarrow z_{ij} = 1$ for all $i, j = 1, \ldots, 4$ defines the total and injective function

$$g(\langle x_i, j\rangle) = \langle z_{ij}, 1\rangle \text{ for all } i, j \in \{1, 2, 3, 4\}.$$

We first transform the stated constraints in C_X. The incompatible assignments for the diagonal constraints in M have the form $\{\langle x_i, k\rangle, \langle x_j, k \pm (i-j)\rangle\}$ for all $i, j, k \in \{1, 2, 3, 4\}$, $i < j$, and $1 \leq k \pm (i-j) \leq 4$. Hence, the induced incompatible assignments are $\{\langle z_{ik}, 1\rangle, \langle z_{j,k\pm(i-j)}, 1\rangle\}$. For example, the diagonal constraint $|x_1 - x_2| \neq 2 - 1$ generates the following incompatible assignments:

$$\{\langle z_{11}, 1\rangle, \langle z_{22}, 1\rangle\}, \{\langle z_{12}, 1\rangle, \langle z_{23}, 1\rangle\}, \{\langle z_{13}, 1\rangle, \langle z_{24}, 1\rangle\},$$
$$\{\langle z_{12}, 1\rangle, \langle z_{21}, 1\rangle\}, \{\langle z_{13}, 1\rangle, \langle z_{22}, 1\rangle\}, \{\langle z_{14}, 1\rangle, \langle z_{23}, 1\rangle\}$$

for inclusion in S_Z. The incompatible assignments for the column constraints in M have the form $\{\langle x_i, k\rangle, \langle x_j, k\rangle\}$ for all $i, j, k \in \{1, 2, 3, 4\}$ and $i < j$. Hence, the induced incompatible assignments are $\{\langle z_{ik}, 1\rangle, \langle z_{jk}, 1\rangle\}$. For example, the constraint $x_1 \neq x_2$ generates the incompatible assignments $\{\langle z_{11}, 1\rangle, \langle z_{21}, 1\rangle\}$, $\{\langle z_{12}, 1\rangle, \langle z_{22}, 1\rangle\}$, $\{\langle z_{13}, 1\rangle, \langle z_{23}, 1\rangle\}$, and $\{\langle z_{14}, 1\rangle, \langle z_{24}, 1\rangle\}$ for inclusion in S_Z.

The No-Double-Assignments constraints for (Z, D_Z) include incompatible assignments transformed from the implicit constraints that each $x_i \in X$ cannot be assigned two different values. Thus:

$$\begin{aligned} N_Z &= \bigcup_{x_i \in X} \{\{g(\langle x_i, j_1\rangle), g(\langle x_i, j_2\rangle)\} \mid j_1, j_2 \in \{1, \ldots, 4\} \\ &\qquad \wedge j_1 < j_2\} \\ &= \{\{\langle z_{ij_1}, 1\rangle, \langle z_{ij_2}, 1\rangle\} \mid i, j_1, j_2 \in \{1, \ldots, 4\} \wedge j_1 < j_2\} \end{aligned}$$

For example, the implicit requirement for $x_1 \in X$ will generate the following incompatible assignments $\{\langle z_{11}, 1\rangle, \langle z_{12}, 1\rangle\}$, $\{\langle z_{11}, 1\rangle, \langle z_{13}, 1\rangle\}$, $\{\langle z_{11}, 1\rangle, \langle z_{14}, 1\rangle\}$, $\{\langle z_{12}, 1\rangle, \langle z_{13}, 1\rangle\}$, $\{\langle z_{12}, 1\rangle, \langle z_{14}, 1\rangle\}$, and $\{\langle z_{13}, 1\rangle, \langle z_{14}, 1\rangle\}$ to ensure that no more than one queen will be placed on row i of the chessboard.

Last but not least, we need to take care of the At-Least-One-Assignment constraints A_Z, which are obtained from the implicit constraints "each $x_i \in X$ must be assigned at least one value" in M. Applying g to the assignments $\langle x_i, 1\rangle, \ldots, \langle x_i, 4\rangle$ for each $x_i \in X$ suggests that the incompatible assignments in (Z, D_Z) are among variables $z_{i1}, \ldots, z_{i4}$. The incompatible assignments are those $\{\langle z_{i1}, q_1\rangle, \ldots, \langle z_{i4}, q_4\rangle\}$ such that $q_1 \neq 1, \ldots, q_4 \neq 1$. Since the domain of all variables z_{ij} is only $\{0, 1\}$, $\{\langle z_{i1}, 0\rangle, \ldots, \langle z_{i4}, 0\rangle\}$ is the only incompatible assignment needed. Thus:

$$\begin{aligned} A_Z = \{&\{\langle z_{11}, 0\rangle, \langle z_{12}, 0\rangle, \langle z_{13}, 0\rangle, \langle z_{14}, 0\rangle\}, \\ &\{\langle z_{21}, 0\rangle, \langle z_{22}, 0\rangle, \langle z_{23}, 0\rangle, \langle z_{24}, 0\rangle\}, \\ &\{\langle z_{31}, 0\rangle, \langle z_{32}, 0\rangle, \langle z_{33}, 0\rangle, \langle z_{34}, 0\rangle\}, \\ &\{\langle z_{41}, 0\rangle, \langle z_{42}, 0\rangle, \langle z_{43}, 0\rangle, \langle z_{44}, 0\rangle\}\}. \end{aligned}$$

The intuitive meaning of these incompatible assignments is that there cannot be no queens in row i of the chessboard.

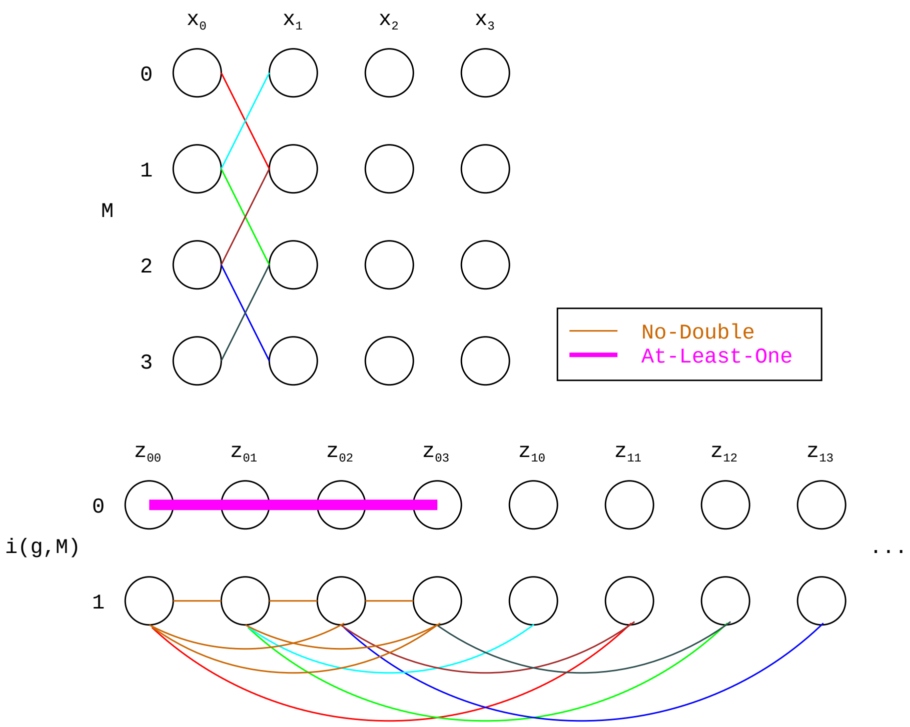


Figure 4.1: The Constraint $|x_0 - x_1| \neq 1$ in M and its Induced Counterpart in $i(g, M)$ with the No-Double-Assignment and At-Least-One-Assignment Constraints from x_0

The induced model $i(g, M) = ((Z, D_Z), C_Z)$ can be formed by extracting and grouping incompatible assignments of the same signatures to form constraints in C_Z. By Theorem 4.1, $i(g, M)$ and M are mutually redundant, and are both models of the 4-queens problem. Figure 4.1 shows the constraint $|x_0 - x_1| \neq 1$ in M and its induced counterpart in $i(g, M)$ with the No-Double-Assignment and At-Least-One-Assignment constraints transformed from the variable x_0. In the figure, the nodes are the assignments and the edges are the incompatible assignments between two nodes. In particular, the edge for the At-Least-One-Assignment constraint is a hyper-edge among the variables z_{00}, z_{01}, z_{02}, and z_{03}. $\square$

Example 4.5 Besides the viewpoint (Z, D_Z), we have another viewpoint for the 4-queens problem for model induction. We consider another viewpoint (Y, D_Y) with four variables $Y = \{y_1, y_2, y_3, y_4\}$ and associated domain function D_Y. Each y_i denotes the row position of the queen on column i and $D_Y(y_i) = \{1, 2, 3, 4\}$ for $i \in \{1, 2, 3, 4\}$. The set of channeling constraints $x_i = j \Leftrightarrow y_j = i$ for all $i, j \in \{1, \ldots, 4\}$ defines the total and injective function

$$f(\langle x_i, j\rangle) = \langle y_j, i\rangle \text{ for all } i, j \in \{1, 2, 3, 4\}.$$

Again, we first transform the stated constraints in C_X. Recall that the incompatible assignments for the diagonal constraints in M have the form $\{\langle x_i, k\rangle, \langle x_j, k\pm(i-j)\rangle\}$ for all $i, j, k \in \{1, 2, 3, 4\}, i < j$, and $1 \leq k\pm(i-j) \leq 4$. Hence, the induced incompatible assignments are $\{\langle y_k, i\rangle, \langle y_{k\pm(i-j)}, j\rangle\}$. For example, the constraint $|x_1 - x_2| \neq 2 - 1$ generates the incompatible assignments $\{\langle y_1, 1\rangle, \langle y_2, 2\rangle\}$, $\{\langle y_2, 1\rangle, \langle y_3, 2\rangle\}$, $\{\langle y_3, 1\rangle, \langle y_4, 2\rangle\}$, $\{\langle y_2, 1\rangle, \langle y_1, 2\rangle\}$, $\{\langle y_3, 1\rangle, \langle y_2, 2\rangle\}$, and $\{\langle y_4, 1\rangle, \langle y_3, 2\rangle\}$ for inclusion in S_Y. The incompatible assignments for the column constraints in M have the form $\{\langle x_i, k\rangle, \langle x_j, k\rangle\}$ for all $i, j, k \in \{1, 2, 3, 4\}$ and $i < j$. Hence, we collect the induced assignment sets $\{\langle y_k, i\rangle, \langle y_k, j\rangle\}$. However, these assignment sets do not form incompatible assignments because they contain two different assignments for the same variable y_k. Therefore, there are no incompatible assignments for inclusion in S_Y corresponding to the column constraints.

The No-Double-Assignments constraints for (Y, D_Y) include incompatible assignments transformed from the implicit constraints that each $x_i \in X$ cannot be assigned two different values. Thus:

$$\begin{aligned} N_Y &= \bigcup_{x_i \in X} \{\{f(\langle x_i, j_1\rangle), f(\langle x_i, j_2\rangle)\} \mid j_1, j_2 \in \{1, \ldots, 4\} \\ &\qquad \wedge j_1 < j_2\} \\ &= \{\{\langle y_{j_1}, i\rangle, \langle y_{j_2}, i\rangle\} \mid i, j_1, j_2 \in \{1, \ldots, 4\} \wedge j_1 < j_2\} \end{aligned}$$

For example, the implicit requirement for $x_1 \in X$ will generate the following incompatible assignments $\{\langle y_1, 1\rangle, \langle y_2, 1\rangle\}$, $\{\langle y_1, 1\rangle, \langle y_3, 1\rangle\}$, $\{\langle y_1, 1\rangle, \langle y_4, 1\rangle\}$,

$\{\langle y_2, 1\rangle, \langle y_3, 1\rangle\}$, $\{\langle y_2, 1\rangle, \langle y_4, 1\rangle\}$, and $\{\langle y_3, 1\rangle, \langle y_4, 1\rangle\}$ to ensure that no more than one queen will be placed on row i of the chessboard.

Last but not least, we need to take care of the At-Least-One-Assignment constraints A_Y, which are obtained from the implicit constraints "each $x_i \in X$ must be assigned at least one value" in M. Applying f to the assignments $\langle x_i, 1\rangle, \ldots, \langle x_i, 4\rangle$ for each $x_i \in X$ suggests that the incompatible assignments in (Y, D_Y) are among variables $y_1, \ldots, y_4$. The incompatible assignments are those $\{\langle y_1, q_1\rangle, \ldots, \langle y_4, q_4\rangle\}$ such that $q_1 \neq 1, \ldots, q_4 \neq 1$. Since the domain of all variables y_i is only $\{1, 2, 3, 4\}$, we have the incompatible assignments $\{\langle y_1, q_1\rangle, \ldots, \langle y_4, q_4\rangle\}$ for all $q_1, q_2, q_3, q_4 \in \{2, 3, 4\}$. The intuitive meaning of these incompatible assignments is that there cannot be no queens in each row of the chessboard. However, the No-Double-Assignment constraints ensure that at most one queen can be placed in each of the four rows, and we have to place one queen in each of the four columns. Therefore, there can never be no queens in a row of the chessboard, and the information from these incompatible assignments is implied already. Hence, we can ignore the At-Least-One-Assignment constraints in this special case and allow $A_Y = \emptyset$.

The induced model $i(f, M) = ((Y, D_Y), C_Y)$ can be formed by extracting and grouping incompatible assignments of the same signatures to form constraints in C_Y. By Theorem 4.1, $i(g, M)$, $i(f, M)$, and M are mutually redundant, and are both models of the 4-queens problem. Figure 4.2 shows the constraint $|x_1 - x_0| \neq 1$ in M and its induced counterpart in $i(f, M)$ with the No-Double-Assignment constraints transformed from the variable x_0. $\square$

4.3 Properties

The following theorem and corollaries give an important consequence of model induction: the transformation of incompatible assignments is meaning-preserving.

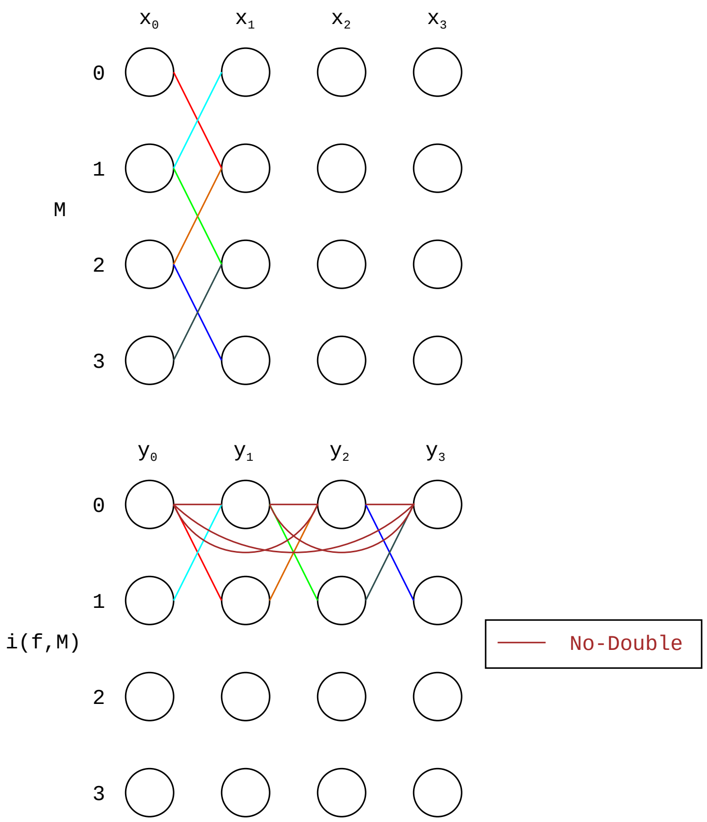


Figure 4.2: The Constraint $|x_0 - x_1| \neq 1$ in M and its Induced Counterpart in $i(f, M)$ with the No-Double-Assignment Constraints from x_0

Theorem 4.1 If $M = (V_1, C_1)$ is a model for problem P, and V_2 is a proper viewpoint of P, then M and $i(f, M)$ are mutually redundant models for all total and injective functions f (defined by channeling constraints connecting V_1 and V_2) mapping from possible assignments in V_1 to those in V_2.

Proof 4.1 Suppose there is a set of channeling constraints defining a total and injective function f such that M and $i(f, M)$ are not mutually redundant. By definition, there is no one-one mapping between $sol(M)$ and $sol(i(f, M))$ and thus also between $sol(P)$ and $sol(i(f, M))$ (since M is a model for P). Thus no subset S of the set of all possible complete assignments in V_2 can have a one-one mapping with $sol(i(f, M))$. Consequently, V_2 is not a proper viewpoint of P; hence a contradiction. □

Corollary 4.2 If $M_1 = (V_1, C_1)$ and $M_2 = (V_2, C_1)$ are mutually redundant models of P, and f is a total and injective function mapping from possible assignments in V_1 to those in V_2, then $sol(M_2) = sol(i(f, M_1))$.

Proof 4.2 By Theorem 4.1, M_1 and $i(f, M_1)$ are mutually redundant. So are M_1, M_2, and $i(f, M_1)$. By definition, there must be a one-one mapping between $sol(M_2)$ and $sol(i(f, M_1))$. Since M_2 and $i(f, M_1)$ share the same viewpoint V_2, the only possible one-one mapping is the identity mapping for V_2 to be proper. Thus $sol(M_2) = sol(i(f, M_1))$. □

Corollary 4.3 If $M = (V_1, C_1)$ is a model for problem P, V_2 is a proper viewpoint of P, and f is a total and bijective function (i.e., f^{-1} exists) mapping from possible assignments in V_1 to those in V_2, then $sol(i(f^{-1}, i(f, M))) = sol(M)$.

Proof 4.3 By Theorem 4.1, M and $i(f, M)$ are mutually redundant. Similarly, $i(f, M)$ and $i(f^{-1}, i(f, M))$ are mutually redundant. M and $i(f^{-1}, i(f, M))$ share the same viewpoint. By Corollary 4.2, $sol(i(f^{-1}, i(f, M))) = sol(M)$. □

Applying model induction twice on Permutation CSPs using channeling constraints f and its inverse f^{-1}, where $f(\langle x_i, j\rangle) = \langle y_j, i\rangle$ for all i, j, can be idempotent. In a *Permutation CSP* [13, 36, 37] $((X, D_X), C)$, we always have $D_X(x_i) = D_X(x_j)$ for all $x_i, x_j \in X$, and $|D_X(x_i)| = |X|$. In addition, any solution $\{\langle x_1, k_1\rangle, \ldots, \langle x_n, k_n\rangle\}$ of a Permutation CSP must have the property that $k_i \neq k_j \Leftrightarrow i \neq j$.

Theorem 4.4 If the followings are true:

- $M = ((X, D), C)$ is a binary Permutation CSP (a binary CSP as well as a Permutation CSP);
- No two constraints in M have the same signature;
- The all-different constraints are ensured by the incompatible assignments $\{\langle x_i, k\rangle, \langle x_j, k\rangle\}$ for all $x_i, x_j \in X$ and $k \in D(x_i)$ with $i \neq j$; and
- We have another viewpoint connected with channeling constraints defining $f(\langle x_i, j\rangle) = \langle y_j, i\rangle$ for all i, j,

then

$$M = i(f^{-1}, i(f, M)).$$

Proof 4.4 Since no two constraints in M have the same signature, we can consider the set of incompatible assignments of all constraints in M. Let $\Theta = \{\theta | c \in C \wedge \theta \in rel(c)\}$ or $\Theta = \Theta_s \cup \Theta_{ad}$ where

$$\Theta_s = \{\theta | c \in C \wedge \theta = \{\langle x_i, j\rangle, \langle x_k, l\rangle\} \in rel(c) \wedge i \neq k \wedge j \neq l\}$$

and

$$\Theta_{ad} = \{\theta | c \in C \wedge \theta \in rel(c) \wedge \theta = \{\langle x_i, k\rangle, \langle x_j, k\rangle\} \wedge i \neq j\}.$$

Θ_{ad} is the set of the incompatible assignments of the all-different constraints, while Θ_s is the set of other incompatible assignments of the model. Similarly, let Θ' and Θ'' be the sets of all incompatible assignments of $i(f, M)$ and $i(f^{-1}, i(f, M))$ respectively.

$$
\begin{aligned}
\Theta' &= \{f(\theta)|\theta \in \Theta_s\} \cup N_Y \\
\Theta'' &= \{f^{-1}(\theta)|\theta \in \Theta' \setminus N_Y\} \cup N_X \\
&= \{f^{-1}(f(\theta))|\theta \in \Theta_s\} \cup N_X \\
&= \{\theta|\theta \in \Theta_s\} \cup \{\{\langle x_i, k\rangle, \langle x_j, k\rangle\}|x_i, x_j \in X \wedge k \in D(x_i) \wedge i \neq j\} \\
&= \Theta_s \cup \Theta_{ad} \\
&= \Theta
\end{aligned}
$$

Since M and $i(f^{-1}, i(f, M))$ have the same set of incompatible assignments, and both of them do not have two constraints with the same signature, therefore, $M = i(f^{-1}, i(f, M))$. □

Chapter 5

Exploiting Redundancy from Model Induction

In this chapter, we focus our interest on induced models which are mutually redundant to their original models. We introduce two operators, namely model intersection and channeling, as two ways of combining models. We identify three new forms of redundancy by using model intersection and/or channeling to combine mutually redundant models. The extra redundant information in the combined models can enhance constraint propagation and hence solving efficiency. We also present experimental results on combining mutually redundant models using our proposed scheme.

5.1 Combining Redundant Models

In this section, we introduce two operators, namely model intersection and channeling, to combine models. Model intersection allows combining two models in the same viewpoint, while model channeling allows combining two models in different viewpoints. For each of the two operators, we give its syntactic construction rule and define its set-theoretic meaning. These operators are useful in combining models to enhance constraint propagation for solving CSPs.

5.1.1 Model Intersection

Model intersection [21] forms *conjuncted models* by essentially conjoining constraints from constituent models. A solution of a conjuncted model must thus also be a solution of all of its constituent models.

Given two models $M_1 = ((X_1, D_{X_1}), C_{X_1})$ and $M_2 = ((X_2, D_{X_2}), C_{X_2})$, the viewpoint $V = (X, D_X)$ of the conjuncted model contains variables from both viewpoints, i.e., $X = X_1 \cup X_2$. If a variable x appears only in either X_1 or X_2, then the domain of x remains the same in D_X; otherwise, $D_X(x)$ is the *intersection* of $D_{X_1}(x)$ and $D_{X_2}(x)$. The constraint set $C_{X_1 \cup X_2}$ is the union of C_{X_1} and C_{X_2}. More formally, the conjuncted model $M_1 \cap M_2$ is $((X, D_X), C_{X_1} \cup C_{X_2})$, where $X = X_1 \cup X_2$ and for all $x \in X$,

$$D_X(x) = \begin{cases} D_{X_1}(x) & \text{if } x \in X_1 \wedge x \notin X_2 \\ D_{X_2}(x) & \text{if } x \notin X_1 \wedge x \in X_2 \\ D_{X_1}(x) \cap D_{X_2}(x) & \text{otherwise} \end{cases}$$

We overload the $\cap$ operator so that it operates on CSP models as well as sets.

Example 5.1 Suppose we have two models $M_1 = ((X_1, D_{X_1}), C_{X_1})$, where

- $X_1 = \{x_1, x_2, x_3\}$ and $D_{X_1}(x_1) = D_{X_1}(x_2) = D_{X_1}(x_3) = \{1, 2, 3\}$,
- $C_{X_1} = \{x_1 \neq x_3\}$,

and $M_2 = ((X_2, D_{X_2}), C_{X_2})$, where

- $X_2 = \{x_2, x_3, x_4\}$ and $D_{X_2}(x_2) = D_{X_2}(x_3) = D_{X_2}(x_4) = \{1, 2\}$,
- $C_{X_2} = \{x_2 = x_3\}$.

Then we have as the conjuncted model $M_1 \cap M_2 = ((X, D_X), C_X)$, where

- $X = \{x_1, \ldots, x_4\}$ and $D_X(x_1) = \{1, 2, 3\}, D_X(x_2) = D_X(x_3) = D_X(x_4) = \{1, 2\}$,

- $C_X = \{x_1 \neq x_3, x_2 = x_3\}$.

$\square$

A consequence of the definition is that every solution of a conjuncted model must satisfy all constraints in its constituent models.

Theorem 5.1 If V is the viewpoint of $M_1 \cap M_2$, then $sol(M_1 \cap M_2) = \{\theta_1 \cup \theta_2 | \theta_1 \in sol(M_1) \wedge \theta_2 \in sol(M_2) \wedge cmpd(\theta_1 \cup \theta_2, V)\}$.

Proof 5.1 Let θ_1 and θ_2 be complete assignments of M_1 and M_2 respectively.

$$\begin{aligned}
& \theta_1 \in sol(M_1) \wedge \theta_2 \in sol(M_2) \wedge cmpd(\theta_1 \cup \theta_2, V) \\
\Leftrightarrow\ & \forall c_1 \in C_{X_1} \cdot [c_1\theta_1] \wedge \forall c_2 \in C_{X_2} \cdot [c_2\theta_2] \wedge cmpd(\theta_1 \cup \theta_2, V) \\
\Leftrightarrow\ & \forall c_1 \in C_{X_1} \cdot [c_1(\theta_1 \cup \theta_2)] \wedge \forall c_2 \in C_{X_2} \cdot [c_2(\theta_1 \cup \theta_2)] \wedge cmpd(\theta_1 \cup \theta_2, V) \\
\Leftrightarrow\ & \forall c \in C_X \cdot [c(\theta_1 \cup \theta_2)] \wedge cmpd(\theta_1 \cup \theta_2, V) \\
\Leftrightarrow\ & \theta_1 \cup \theta_2 \in sol(M_1 \cap M_2)
\end{aligned}$$

$\square$

Theorem 5.1 gives a construction of the solution set of $M_1 \cap M_2$ from those of M_1 and M_2. In particular, solutions θ_1 of M_1 and θ_2 of M_2 can be combined to form a solution of $M_1 \cap M_2$ if and only if each shared variable in θ_1 and θ_2 is assigned the same value in θ_1 and θ_2. Otherwise, $\theta_1 \cup \theta_2$ cannot form a compound assignment in the conjuncted viewpoint. This condition is enforced by the *cmpd* predicate.

When two models $M_1 = (V, C_1)$ and $M_2 = (V, C_2)$ share the same viewpoint, $M_1 \cap M_2 = (V, C_1 \cup C_2)$. Furthermore, if M_1 and M_2 are mutually redundant, then $sol(M_1) = sol(M_2) = sol(M_1 \cap M_2)$. This property allows us to combine two models of the same problem in the same viewpoint with the conjuncted model still having the same solution set as the individual models.

5.1.2 Model Channeling

Suppose there is a set C_c of channeling constraints connecting the viewpoints V_1 and V_2. *Model channeling* [8, 21] combines M_1 and M_2 using C_c to form a *channeled model*, which is $M_1 \cap M_2$ plus the channeling constraints C_c. More formally, the channeled model $M_1 \overset{C_c}{\bowtie} M_2$ is $((X, D_X), C_{X_1} \cup C_{X_2} \cup C_c)$, where $X = X_1 \cup X_2$ and for all $x \in X$,

$$D_X(x) = \begin{cases} D_{X_1}(x) & \text{if } x \in X_1 \wedge x \notin X_2 \\ D_{X_2}(x) & \text{if } x \notin X_1 \wedge x \in X_2 \\ D_{X_1}(x) \cap D_{X_2}(x) & \text{otherwise} \end{cases}$$

Given two models M_1 and M_2. The channeled model $M_1 \overset{C_c}{\bowtie} M_2$ is more constrained than the conjuncted model $M_1 \cap M_2$. A solution of $M_1 \overset{C_c}{\bowtie} M_2$ must satisfy all constraints in M_1 and M_2 plus the channeling constraints C_c.

Theorem 5.2 If V is the viewpoint of $M_1 \overset{C_c}{\bowtie} M_2$, then $sol(M_1 \overset{C_c}{\bowtie} M_2) = \{\theta_1 \cup \theta_2 | \theta_1 \in sol(M_1) \wedge \theta_2 \in sol(M_2) \wedge cmpd(\theta_1 \cup \theta_2, V) \wedge \forall c \in C_c \cdot [c(\theta_1 \cup \theta_2)\}$.

Proof 5.2 Similar to the proof of Theorem 5.1. □

The redundant modeling approach by Cheng *et al.* [8] is a special case of model channeling.

Theorem 5.3 Suppose M_1 and M_2 are mutually redundant models. If C_c is a set of channeling constraints enforcing the one-one mapping between $sol(M_1)$ and $sol(M_2)$, then M_1, M_2, and $M_1 \overset{C_c}{\bowtie} M_2$ are mutually redundant to one another.

Proof 5.3 By definition, any solution θ of $M_1 \overset{C_c}{\bowtie} M_2$ can be projected to V_1 and V_2 to form solutions of M_1 and M_2 respectively. Suppose θ_1 is a solution of M_1. Then there must be θ_2 that can be mapped to θ_1 by C_c (and *vice versa*) and is a solution of M_2. In addition, $cmpd(\theta_1 \cup \theta_2, V)$, where V is the viewpoint of $M = M_1 \overset{C_c}{\bowtie} M_2$, must hold since $\theta_1 \cup \theta_2$ satisfies C_c and shared variables in θ_1 and θ_2 must share the same assignments. Therefore, there is a one-one mapping between $sol(M_1)$ and $sol(M)$. Similarly, we can show the same between $sol(M_2)$ and $sol(M)$. □

5.2 Three New Forms of Model Redundancy

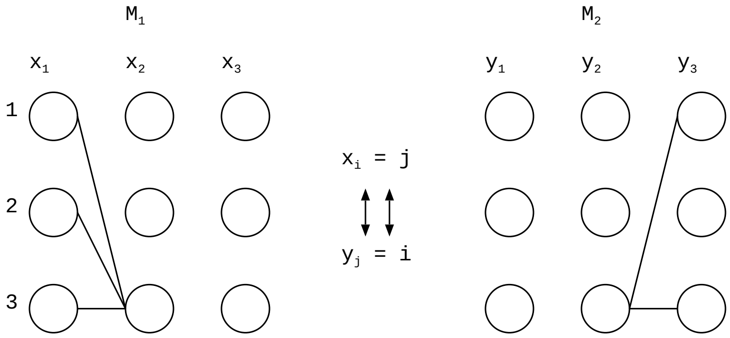
A viewpoint can greatly influence how a human modeler looks at a problem. Each viewpoint provides a distinct perspective emphasizing perhaps a specific aspect of the problem. Therefore, the modeler will likely express individual constraints differently under different viewpoints, although the constraints under each viewpoint should collectively give the same solutions to the problem being modeled. In particular, a constraint expressed for one viewpoint might not even have an (explicit) counterpart in the other viewpoint, and *vice versa* [25].

Suppose $M_1 = (V_1, C_1)$ and $M_2 = (V_2, C_2)$ are mutually redundant models with different viewpoints handcrafted by human modeler. We also have a set C_c of channeling constraints defining a total and injective function f from possible assignments of V_1 to those of V_2. Model induction essentially translates constraint information expressed in V_1 to V_2 via channeling constraints f. The transformed constraints express in V_2 the constraint information of the problem as viewed from V_1. These transformed constraints are likely different from constraints expressed directly using V_2 by the human modeler. Therefore, $i(f, M_1)$ and M_2 are redundant and yet complementary to each other.

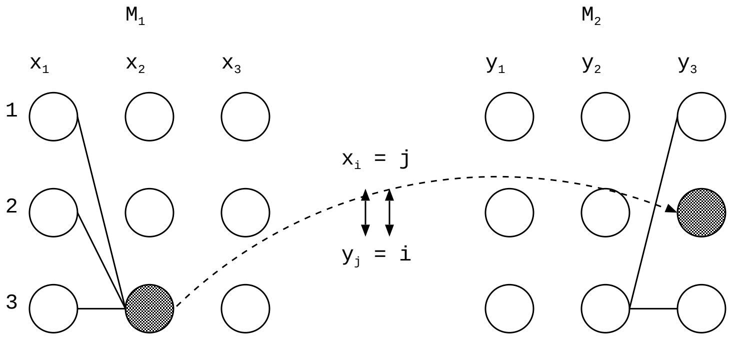
Model channeling and intersection give various possibilities to combine M_1, M_2, and models induced from the two models. Model channeling is "collaborative" in nature. It allows each sub-model to perform constraint propagation on its own, and yet communicate its results (variable instantiation and domain pruning) to the other sub-models to possibly initiate further constraint propagation. Furthermore, model channeling allows constraint propagation to explore different variable spaces (viewpoints). For example, Figure 5.1(a) shows part of two models being channeled with the channeling constraints $x_i = j \Leftrightarrow y_j = i$ for all i, j pairs. Constraint propagation in M_1 would prune the value 3 from $D_X(x_2)$ but no more. Since M_1 and M_2 are channeled through the channeling constraints, the pruning of $\langle x_2, 3\rangle$ would cause $\langle y_3, 2\rangle$ to be pruned also, resulting in the state in Figure 5.1(b). Now, the pruning of $\langle y_3, 2\rangle$ makes $y_2 = 3$ inconsistent with y_3, so $\langle y_2, 3\rangle$ in M_2 and hence $\langle x_3, 2\rangle$ in M_1 are pruned. Figure 5.1(c) shows the final state of propagation. It can be seen that the channeled model has increased propagation over M_1 or M_2 alone.

Model intersection is "additive" in that it merges constraints to form stronger constraints, which is the source of increased constraint propagation. For example, suppose we have a constraint $x > 1$ in M_1 and another constraint $x < 3$ in M_2, with $D_X(x) = \{0, \ldots, 4\}$ in both M_1 and M_2. Constraint propagation in M_1 would prune the values 0 and 1 from $D_X(x)$. Similarly, constraint propagation in M_2 would prune the values 3 and 4 from $D_X(x)$. In the conjuncted model $M_1 \cap M_2$, the pruning information of M_1 and M_2 is combined and x would be automatically assigned the value 2, and this assignment may initiate further propagation.

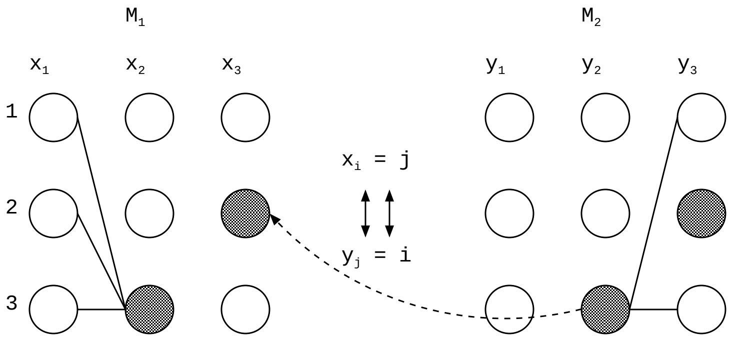
The benefits of model intersection may not seem to be obvious with examples using unary constraints because they can be preprocessed by node consistency techniques before searching and discarded afterwards. Therefore, we give another example with binary constraints. Suppose we have a constraint $x + y < 3$ in M_1 and another constraint $x - y > 1$ in M_2, with $D(x) = D(y) = \{0, \ldots, 4\}$. Constraint propagation in M_1 would prune the values 3 and 4 from $D(x)$ and $D(y)$. Similarly, constraint propagation in M_2 would prune the values 0 and 1 from $D(x)$ and the values 3 and 4 from $D(y)$. In the conjuncted model $M_1 \cap M_2$, these pruning information is combined and,



(a) Initial State



(b) Intermediate State



(c) Final State

Figure 5.1: Constraint Propagation on Model Channeling between Two Models

x would be automatically assigned the value 2. The assignment $\langle x, 2\rangle$ would cause all $j \in D(y)$ with $j < 1$ be removed because both $2+y < 3$ and $2-y > 1$ imply $y < 1$. Hence, $D(y) = \{0\}$ and y would be automatically assigned the value 0, and the assignment $\langle y, 0\rangle$ may initiate further propagation. This example shows that the conjuncted model can prune more than each of the individual models can do, and hence model intersection is beneficial to CSP solving.

Assuming f^{-1} *exists*, we propose three classes of interesting combined models.

- $i(f, M_1) \cap M_2$ and $M_1 \cap i(f^{-1}, M_2)$
- $M_1 \overset{C_c}{\bowtie} i(f, M_1)$ and $M_2 \overset{C_c}{\bowtie} i(f^{-1}, M_2)$
- $(i(f, M_1) \cap M_2) \overset{C_c}{\bowtie} (i(f^{-1}, i(f, M_1) \cap M_2))$ and $(i(f^{-1}, M_2) \cap M_1) \overset{C_c}{\bowtie} (i(f, i(f^{-1}, M_2) \cap M_1))$

We note that f^{-1} always exists for Permutation CSPs. Therefore, we can always perform model induction using either M_1 or M_2.

5.3 Experiments

To verify the feasibility and efficiency of our proposal, we realize and evaluate various models of several problems using ILOG Solver 4.4 [18] running on a Sun Ultra 5/400 workstation with 256M of memory. These problems include the Langford's problem, random Permutation CSPs, Golomb rulers and circular Golomb rulers problem, and all-interval series problem. All of them can be modeled as Permutation CSPs and hence we can make use of model induction to produce their induced counterparts. We use the `IlcTableConstraint` function [18] to create constraints from sets of incompatible assignments. The channeling constraints are enforced using the `IlcInverse` constraint, which is essentially equivalent to the set of constraints $x_i = j \Leftrightarrow y_j = i$ for all i, j pairs. Full arc consistency or generalized arc consistency are enforced in constraint propagation. Variables are chosen using the smallest-domain-first variable-ordering heuristic, which is to choose the variable with the smallest domain size at each level of the search tree. Ties are broken with a predefined order on the variables. This variable ordering heuristic minimizes the branching factor of the search tree, and hence is a good heuristic in many situations.

In a Permutation CSP, we have to use all-different constraints to ensure that a solution forms a permutation. There are efficient generalized arc consistency algorithms devoted to the all-different constraints which can potentially remove more domain values during search. However, building a set of incompatible assignments for the GAC all-different constraints involves all the variables in a model. The process is time-consuming and the size of the set is also large. Therefore, we simply use a set of disequality constraints between pairwise variables in all our models instead of enforcing GAC.

When intersecting two models, we have the option of *merging constraints* with the same signature into one constraint by taking the union of the constraints' sets of incompatible assignments. For example, suppose $sig(c_1) = sig(c_2)$, we can construct a merged constraint c' to replace c_1 and c_2 such that $sig(c') = sig(c_1) = sig(c_2)$ and $rel(c') = rel(c_1) \cup rel(c_2)$. The resultant constraint c', having a *more* global view on the variables in $sig(c')$, can potentially provide more constraint propagation than the individual constraints c_1 and c_2 when used separately. For example, suppose $D(x) = D(y) = \{1, 2, 3\}$, and we have two constraints $c_1 : x = 2 \Rightarrow y = 2$ and $c_2 : x \neq y$. Both c_1 and c_2 have signatures $sig(c_1) = sig(c_2) = \{x, y\}$. Figure 5.2 shows the configurations of both constraints and the merged constraint c'. Although both c_1 and c_2 are arc consistent, the merged constraint is not. We can prune the value 2 from $D(x)$ since $\langle x, 2\rangle$ has no support in $D(y)$. The opportunity for constraint merging arises naturally for model intersection since it is quite plausible for two

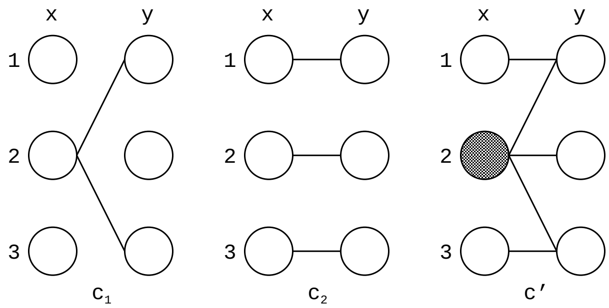


Figure 5.2: Increased Propagation due to Constraint Merging

separate models to have different constraints with the same signature. Note that constraint merging is applicable, not just in the context of model intersection, whenever we have more than one individual constraint with the same signature in a CSP. In our experiments, since we explicitly use incompatible assignments to build constraints, it is straightforward to merge them together using `IlcTableConstraint`. Hence, we merge constraints whenever possible, so that every constraint in a model has a unique signature.

5.3.1 Langford's Problem

The Langford's problem, listed as "prob024" in CSPLib [14], can be modeled as a Permutation CSP having all the desired properties for experimenting with model induction and model channeling. In the Langford's problem, there is a $m \times n$-digit sequence which includes the digits 1 to n, with each digit occurs m times. There is one digit between any consecutive pair of digit 1, two digits between any consecutive pair of digit 2, ..., n digits between any consecutive pair of digit n. The Langford's problem, denoted as (m, n) problem, is to find such a sequence (or all sequences).

Smith [36] suggests two ways to model the the Langford's problem a CSP. We use the $(3, 9)$ instance to illustrate the two models. In the first model M_1, we use 27 variables $X = \{x_0, \ldots, x_{26}\}$, which we can think of as 1_1, 1_2, 1_3, 2_1,

..., 9_2, and 9_3. Here, 1_1 represents the first digit 1 in the sequence, 1_2 represents the second digit 1, and so on. The domains of these variables are the values that represent the positions of a digit in the sequence. We use $\{0, \ldots, 26\}$ to represent the domains. Hence, we have the viewpoint $V_1 = (X, D_X)$, where $D_X(x_i) = \{0, \ldots, 26\}$ for $i \in \{0, \ldots, 26\}$.

Using this viewpoint, we can formulate the problem using two types of constraints. The all-different constraints ensure that all digits are placed in different positions in the sequence, whereas the separation constraints ensure that the spacings between consecutive pair of the same digit are correct.

- all-different constraints: $x_i \neq x_j$ for all $0 \leq i < j \leq 26$
- separation constraints: $x_{i+1} = x_i + 2, x_{i+2} = x_{i+1} + 2$ for all $0 \leq i \leq 8$

In the second model M_2, we again use 27 variables $Y = \{y_0, \ldots, y_{26}\}$ to represent each position in the sequence. Their domains are $\{0, \ldots, 26\}$, whose elements correspond to the digits $1_1, 1_2, 1_3, 2_1, \ldots, 9_2, 9_3$. Hence, we have the viewpoint $V_2 = (Y, D_Y)$, where $D_Y(y_i) = \{0, \ldots, 26\}$ for $i \in \{0, \ldots, 26\}$.

Using this viewpoint, we also have two types of constraints as in V_1:

- all-different constraints: $y_i \neq y_j$ for all $0 \leq i < j \leq 26$
- separation constraints: $y_j = 3i \Leftrightarrow y_{j+(i+2)} = 3i+1, y_j = 3i \Leftrightarrow y_{j+2(i+2)} = 3i+2$ for all valid i, j pairs, and $y_j \neq i$ for all $0 \leq i \leq 8$ and $27-2(i+2) \leq j \leq 26$

This time, the all-different constraints ensure that all positions are occupied by different digits. The constraints $y_j \neq i$ for all $0 \leq i \leq 8$ and $27 - 2(i+2) \leq j \leq 26$ ensure that a digit is not the rightmost in the sequence.

We can write the channeling constraints C_c connecting V_1 and V_2 as $x_i = j \Leftrightarrow y_j = i$ for all $i, j = 0, \ldots, 26$. These constraints define a total and bijective function f where $f(\langle x_i, j \rangle) = \langle y_j, i \rangle$ for all valid i, j. With M_1, M_2, and f, we can construct the three proposed classes of combined models. We note that, for the special case of binary Permutation CSPs, $i(f^{-1}, i(f, M_1) \cap M_2) = M_1 \cap i(f^{-1}, M_2)$ with constraint merging.

Corollary 5.4 If the followings are true:

- $M_1 = ((X, D_X), C_X)$ and $M_2 = ((Y, D_Y), C_Y)$ are Permutation CSPs and M_1 is binary;
- The all-different constraints in M_1 are ensured by the incompatible assignments $\{\langle x_i, k\rangle, \langle x_j, k\rangle\}$ for all $x_i, x_j \in X$ and $k \in D_X(x_i)$ with $i \neq j$; and
- (X, D_X) and (Y, D_Y) are connected with channeling constraints defining $f(\langle x_i, j\rangle) = \langle y_j, i\rangle$ for all i, j,

then

$$i(f^{-1}, i(f, M_1) \cap M_2) = M_1 \cap i(f^{-1}, M_2) \text{ with constraint merging.}$$

Proof 5.4 With constraint merging, we have

$$i(f^{-1}, i(f, M_1) \cap M_2) = i(f^{-1}, i(f, M_1)) \cap i(f^{-1}, M_2).$$

By Theorem 4.4, $i(f^{-1}, i(f, M_1)) = M_1$. Hence,

$$\begin{aligned} i(f^{-1}, i(f, M_1) \cap M_2) &= i(f^{-1}, i(f, M_1)) \cap i(f^{-1}, M_2) \\ &= M_1 \cap i(f^{-1}, M_2) \end{aligned}$$

□

Since both M_1 and M_2 of the Langford's problem are binary Permutation CSPs, by Corollary 5.4, we have:

- $i(f, M_1 \cap i(f^{-1}, M_2)) = i(f, M_1) \cap M_2$ and

- $i(f^{-1}, i(f, M_1) \cap M_2) = M_1 \cap i(f^{-1}, M_2)$

It is thus only necessary to consider

$$(M_1 \cap i(f^{-1}, M_2)) \overset{C_c}{\bowtie} (i(f, M_1) \cap M_2)$$

for the third class of combined models.

The ways to express the separation constraints are different in the two viewpoints. Suppose we want to express the requirement that the first and second occurrences of digit 1 are separated by one position in the $(2, 3)$ instance. We use one constraint $x_1 = x_0 + 2$ in V_1 to express this information. Figure 5.3 shows this constraint in M_1 and its counterpart in $i(f, M_1)$. The same information, however, is represented differently in M_2. We have to use the constraints $y_0 = 0 \Leftrightarrow y_2 = 1$, $y_1 = 0 \Leftrightarrow y_3 = 1$, $y_2 = 0 \Leftrightarrow y_4 = 1$, $y_3 = 0 \Leftrightarrow y_5 = 1$, $y_4 \neq 0$, and $y_5 \neq 0$ to represent the requirement. Figure 5.4 shows these constraints in M_2. We can see that although $i(f, M_1)$ and M_2 share the same viewpoint, the same information is represented differently. Therefore, if we combined these mutually redundant information using model intersection, we have more redundant information to help enhance constraint propagation and hence speed up the solving process. Figure 5.5 shows the constraints in the conjuncted model $i(f, M_1) \cap M_2$.

We evaluate the various models for the $(3, 9), (3, 10), (3, 11), (4, 9), (4, 10)$, and $(4, 11)$ instances of the Langford's problem, in which only the first two instances are satisfiable. Tables 5.1 and 5.2 show our comparison results of finding the soluble and insoluble instances respectively. Column 1 gives the models. In models with more than one viewpoint, it suffices to search/label variables of either viewpoint, although one may choose to search on both. In column 2, we give also the search variables. The remaining columns report the execution results of the instances. In Table 5.1, we report the results for solving only the first solution and solving for all solutions, while in Table 5.2, we report the results for proving that the instances have no solutions. Each

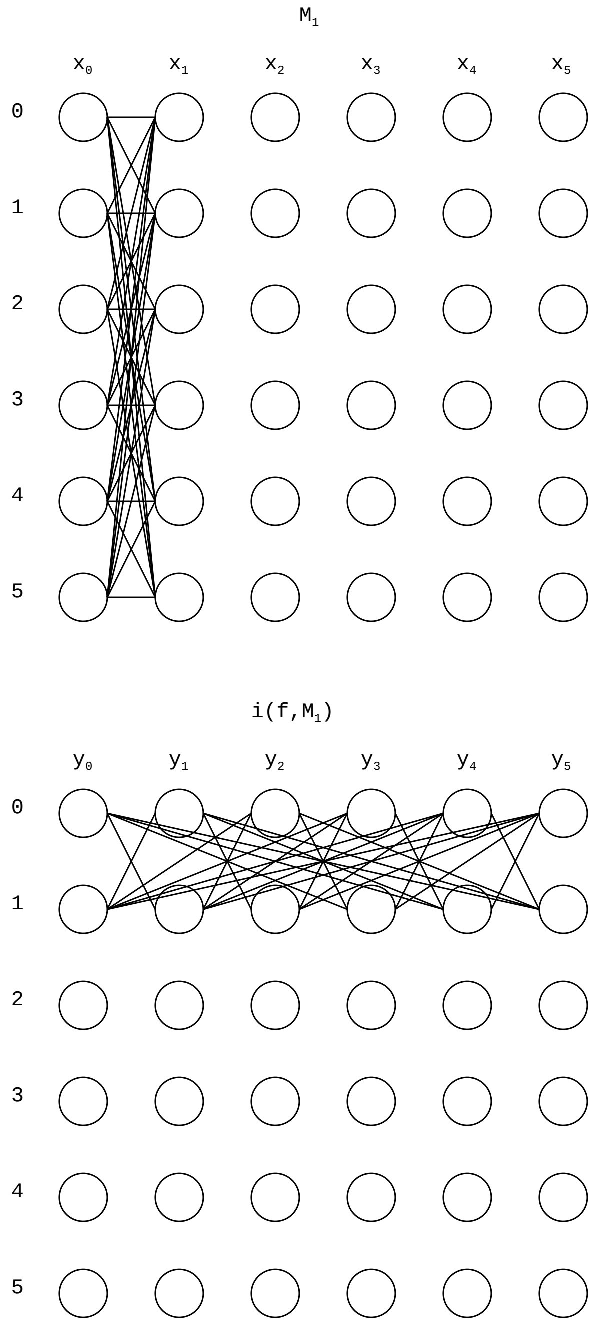


Figure 5.3: The Constraint $x_1 = x_0 + 2$ in M_1 and its Induced Counterpart in $i(f, M_1)$ of the $(2, 3)$ Instance

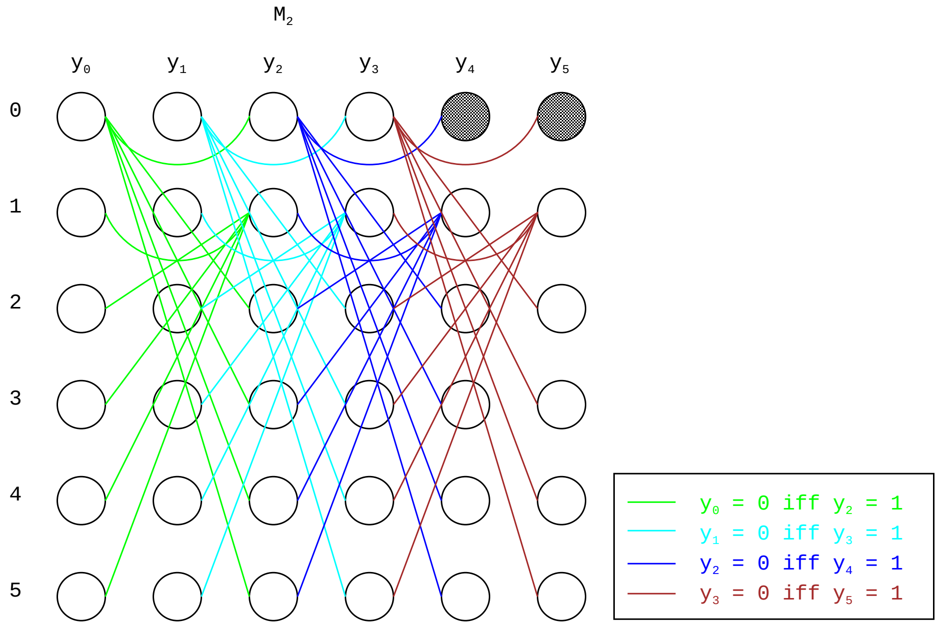


Figure 5.4: The Constraints for Correct Separation of 1_1 and 1_2 in M_2 of the $(2, 3)$ Instance

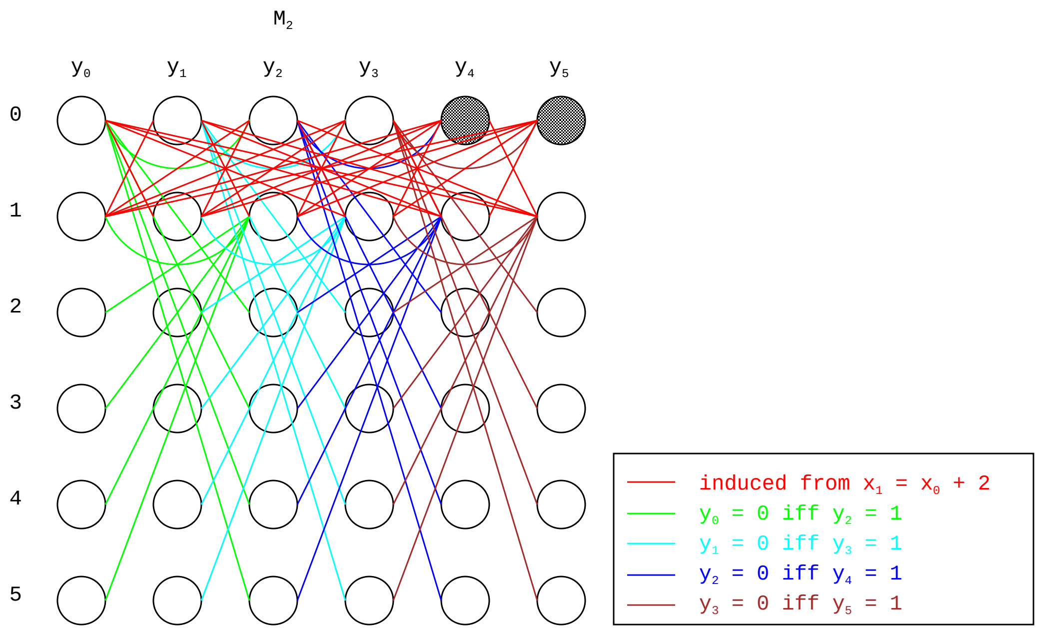


Figure 5.5: The Constraints for Correct Separation of 1_1 and 1_2 in $i(f, M_1) \cap M_2$ of the $(2, 3)$ Instance

cell contains both the number of fails and CPU time in sec (in bracket) of an execution. A cell labeled with "-" means that execution does not terminate within 10 minutes of CPU time. We also highlight in bold the best result of each column.

Each row corresponds to a particular model. We divide the models into five groups, the first two of which are used as control in the experiment. The first group consists of individual models, while the second group consists of combined models constructed using the redundant modeling approach [8]. The remaining groups correspond to our three proposed classes of combined models.

In analyzing the results, attention is sought not just on the CPU time, but also on the number of fails. In fact, the latter is more important and accurate as a measure of the robustness of a model. Combined models are bigger in size, and higher execution overhead is expected. The idea of combining redundant models is to spend more time in constraint propagation in the hope that the extra effort can result in substantial pruning of the search space. A model that gives more pruning has a higher possibility in solving problems that are otherwise computationally infeasible when expressed in weaker models.

The first and fifth groups of models represent the two ends of a spectrum, which indicates the amount of model redundancy utilized in the models. The single models in the first group use no redundancy, and thus performs the worst in terms of the number of fails. Their execution times are not among the worst since these models are the smallest in size, incurring the least execution overhead in constraint propagation. Note also that model M_2 is a poor model. Any model involving M_2 as a base model is bound to perform poorly, both in terms of CPU time and number of fails. In the following, we focus on only models using M_1 as a base model.

The second group makes use of only model channeling, which helps M_1 and M_2 share pruning and variable instantiation information. Constraint propagation also takes place in both viewpoints. Another advantage of this approach

Models	Search Variables	First Solution		All Solutions	
		$(3,9)$	$(3,10)$	$(3,9)$	$(3,10)$
M_1	X	192 (2.43)	569 (8.33)	938 (10.97)	3114 (44.95)
$i(f, M_1)$	Y	-	-	-	-
M_2	Y	-	-	-	-
$i(f^{-1}, M_2)$	X	-	-	-	-
$M_1 \overset{C_c}{\bowtie} M_2$	X	63 (2.75)	193 (8.43)	310 (11.87)	1109 (46.97)
$M_1 \overset{C_c}{\bowtie} M_2$	Y	44 (1.79)	20 (1.63)	322 (11.08)	980 (41.44)
$M_1 \overset{C_c}{\bowtie} M_2$	$X \cup Y$	39 (2.03)	104 (6.08)	225 (9.3)	697 (34.19)
$M_1 \cap i(f^{-1}, M_2)$	X	105 (**1.74**)	313 (5.78)	524 (**7.33**)	1650 (**29.97**)
$i(f, M_1) \cap M_2$	Y	-	-	-	-
$M_1 \overset{C_c}{\bowtie} i(f, M_1)$	X	73 (3.08)	207 (9.81)	401 (14.32)	1221 (55.99)
$M_1 \overset{C_c}{\bowtie} i(f, M_1)$	Y	47 (1.99)	21 (1.72)	338 (12.87)	1021 (48.27)
$M_1 \overset{C_c}{\bowtie} i(f, M_1)$	$X \cup Y$	42 (2.2)	113 (6.76)	239 (10.4)	730 (38.51)
$M_2 \overset{C_c}{\bowtie} i(f^{-1}, M_2)$	X	115 (6.18)	340 (22.33)	561 (29.18)	1812 (121.15)
$M_2 \overset{C_c}{\bowtie} i(f^{-1}, M_2)$	Y	589 (23.98)	475 (23.64)	1462 (59.2)	5547 (279.78)
$M_2 \overset{C_c}{\bowtie} i(f^{-1}, M_2)$	$X \cup Y$	113 (6.04)	338 (22.39)	550 (28.85)	1788 (120.02)
$(M_1 \cap i(f^{-1}, M_2)) \overset{C_c}{\bowtie} (i(f, M_1) \cap M_2)$	X	38 (2.51)	132 (7.53)	195 (10.04)	745 (39.66)
$(M_1 \cap i(f^{-1}, M_2)) \overset{C_c}{\bowtie} (i(f, M_1) \cap M_2)$	Y	31 (1.76)	**9** (**1.56**)	217 (9.82)	665 (35.95)
$(M_1 \cap i(f^{-1}, M_2)) \overset{C_c}{\bowtie} (i(f, M_1) \cap M_2)$	$X \cup Y$	**29** (2.04)	83 (5.95)	**156** (8.6)	**514** (30.9)

Table 5.1: Comparison Results Using (3, 9) and (3, 10) of the Langford's Problem

Models	Variables	$(3, 11)$	$(4, 9)$	$(4, 10)$	$(4, 11)$
M_1	X	14512 (239.07)	138 (7.76)	472 (25.98)	1876 (106.95)
$i(f, M_1)$	Y	-	-	-	-
M_2	Y	-	-	-	-
$i(f^{-1}, M_2)$	X	-	-	-	-
$M_1 \overset{C_c}{\bowtie} M_2$	X	4353 (214.66)	54 (8.06)	185 (26.23)	600 (93.41)
$M_1 \overset{C_c}{\bowtie} M_2$	Y	3657 (175.73)	60 (8.3)	144 (22.77)	384 (66.94)
$M_1 \overset{C_c}{\bowtie} M_2$	$X \cup Y$	2692 (144.98)	53 (7.76)	113 (20.99)	306 (62.06)
$M_1 \cap i(f^{-1}, M_2)$	X	7621 (152.98)	100 (**5.89**)	284 (**17.83**)	1078 (68.57)
$i(f, M_1) \cap M_2$	Y	-	-	-	-
$M_1 \overset{C_c}{\bowtie} i(f, M_1)$	X	4873 (249.23)	58 (9.5)	190 (32.26)	609 (109.53)
$M_1 \overset{C_c}{\bowtie} i(f, M_1)$	Y	3801 (205.44)	61 (10.32)	147 (27.62)	403 (80.97)
$M_1 \overset{C_c}{\bowtie} i(f, M_1)$	$X \cup Y$	2774 (161.6)	54 (9.06)	110 (23.93)	317 (70.9)
$M_2 \overset{C_c}{\bowtie} i(f^{-1}, M_2)$	X	8130 (597.08)	104 (20.58)	328 (72.18)	1160 (288.08)
$M_2 \overset{C_c}{\bowtie} i(f^{-1}, M_2)$	Y	25344 (1448.44)	697 (67.06)	1963 (240.84)	7918 (1127.59)
$M_2 \overset{C_c}{\bowtie} i(f^{-1}, M_2)$	$X \cup Y$	7907 (586.06)	104 (20.61)	328 (72.2)	1155 (288.1)
$(M_1 \cap i(f^{-1}, M_2)) \overset{C_c}{\bowtie} (i(f, M_1) \cap M_2)$	X	2973 (178.81)	44 (7.73)	139 (22.5)	392 (79.8)
$(M_1 \cap i(f^{-1}, M_2)) \overset{C_c}{\bowtie} (i(f, M_1) \cap M_2)$	Y	2553 (151.86)	46 (8.21)	115 (20.99)	287 (61.3)
$(M_1 \cap i(f^{-1}, M_2)) \overset{C_c}{\bowtie} (i(f, M_1) \cap M_2)$	$X \cup Y$	**1937** (**127.77**)	**42** (7.87)	**96** (19.97)	**268** (**58.58**)

Table 5.2: Comparison Results of Proving Insolubility Using $(3, 11)$, $(4, 9)$, $(4, 10)$, and $(4, 11)$ of the Langford's Problem

is that constraints in M_1 and M_2, constructed under different viewpoints, are complementary to each other. These characteristics are the source of increased constraint propagation, and thus drastic cut in the number of fails as compared to the models in the first group.

The third group of models uses only one viewpoint, but model intersection combines the constraints from the two models to form stronger constraints, thus entailing again more constraint propagation. We note, however, that the reduction in the number of fails is not as substantial as the case in the second group of models.

The fourth group of models employs both model induction and model channeling. The models inherit the good characteristics of model channeling, except that the constraints in both models are essentially from M_1. These models are deprived of the chance to share constraint information from M_2. Therefore, the performance of the fourth group is consistently and slightly worse than that of the second group.

The model in the fifth group enjoys the best of both worlds. Each of the sub-models is a conjuncted model, encompassing strengthened constraints obtained from model intersection. The conjuncted models are then connected via model channeling to take advantage of the sharing of pruning information and constraint propagation in different viewpoints. That explains why models in this group always give the lowest number of fails in all benchmarks. Their timings, although not the fastest, are also respectable compared to the fastest time of the respective benchmarks, although these models are the largest in size.

5.3.2 Random Permutation CSPs

A random (binary) CSP (n, m, p_1, p_2) is characterized by four parameters, where n is the number of variables, m is the domain size of the variables.

Constraint density p_1 is the probability that there is a constraint between two variables. Constraint tightness p_2 is the conditional probability that, given a constraint between two variables, a pair of values is inconsistent for the pair of variables.

In a Permutation CSP, the number of variables must be equal to the domain size of the variables. Therefore, we have $n = m$ and a random Permutation CSP (n, p_1, p_2) can be characterized by three parameters. It is generated in two steps. First, we generate a random CSP (n, n, p_1, p_2). Second, we artificially add the incompatible assignments $\{\langle x_i, k\rangle, \langle x_j, k\rangle\}$ for all i, j, k pairs with $i \neq j$ which represent the all-different constraints of the random CSP. Therefore, the actual constraint density in a random Permutation CSP is 1, due to the all-different constraint, but only a portion, i.e., p_1, of them may have incompatible assignments other than those of the all-different constraints. Also, given a constraint between two variables in a random CSP, we note that the expected number of incompatible assignments in the constraint of the random Permutation CSP is $(n^2 - n)p_2 + n$ instead of $n^2 p_2$ in the random CSP.

In a Permutation CSP, a second viewpoint always exists when we interchange the roles of variables and values. While in the original viewpoint, we assign values to variables, in the second viewpoint, we can "assign" variables to values. The channeling constraints connecting these two viewpoint are the same as those in the Langford's problem, i.e., $x_i = j \Leftrightarrow y_j = i$, or $f(\langle x_i, j\rangle) = \langle y_j, i\rangle$ for all i, j pairs. Hence, we can always construct an induced model $i(f, M)$ using f for a Permutation CSP model M. Since M and $i(f, M)$ are mutually redundant, the channeled model $M \overset{C_c}{\bowtie} i(f, M)$ is also mutually redundant to them. In our experiments, we evaluate the models M, $i(f, M)$, and $M \overset{C_c}{\bowtie} i(f, M)$ for the instances of the random Permutation CSPs. In the channeled model, it suffices to label the variables of one model only, or to label both of them. Therefore, we also distinguish the different behaviours of the channeled model due to different labeling strategies.

Cheeseman *et al.* [4] note that many NP-complete problems [9] can be summarized by at least one "order parameter" and that the hard problems occur at a critical value of the parameter. This critical value separates the problem instances into two regions. In one region, most problem instances are easily soluble, whereas in the other region, most problem instances are easily proved insoluble. The transition from one region to another is known as *phase transition*. Smith and Dyer [38] conduct an extensive study on locating the phase transition of random CSPs. Given a set of parameters n, m, p_1, a peak is observed at a critical value of p_2 where the most search efforts are required for finding one solution or proving no solutions exist. According to Smith and Dyer, the expected number of solutions of a random CSP is given by

$$E(N) = m^n(1 - p_2)^{n(n-1)p_1/2}.$$

Since we are considering random Permutation CSPs, we must modify this formula to suit our cases. In a Permutation CSP, we have $m = n$. Besides, we artificially add the incompatible assignments of the all-different constraints to a random CSP to form a random Permutation CSP. The search space is therefore reduced from m^n to $n!$. Hence, the expected number of solutions of a random Permutation CSP is given by

$$E(N) = n!(1 - p_2)^{n(n-1)p_1/2}.$$

Smith and Dyer suggest that setting the expected number of solutions to be 1 would be a good predictor for locating the peak. Similarly in our case, we obtain the predictor

$$\hat{p}_2 = 1 - n!^{-2/(n(n-1)p_1)} \tag{5.1}$$

to locate the peak at phase transition.

In the following, we fix $p_1 = 0.6, 0.8, 1.0$ and vary p_2 in steps of 0.01 up to 1.0. We choose $n = 15$ for finding all solutions and $n = 20$ for finding

n	p_1	$\hat{p}_2$
15	0.6	0.358
15	0.8	0.283
15	1.0	0.233
20	0.6	0.310
20	0.8	0.243
20	1.0	0.200

Table 5.3: Peak Predictors of some Series of Random Permutation CSPs

first solution or proving no solutions exist. The series of experiments with a particular n and p_1 is referred by (n, p_1). Table 5.3 shows the corresponding peak predictors according to formula (5.1). We generate ten instances for each set of parameters and use the median values in plotting the number of fails and CPU time in the graphs.

The top graphs of Figures 5.6, 5.7, and 5.8 show the amount of search in terms of number of fails for finding first solution or proving no solutions exist for the series $(20, 1)$, $(20, 0.8)$, and $(20, 0.6)$ at different p_2 levels respectively. There are three curves for the channeled model, corresponding to searching the variables in the original model, in the induced model, and in both models respectively. The bottom graphs of the figures show the ratio of the number of fails of the induced model or the channeled model to that of the original model. A model with ratio higher (lower) than 1 means it needs more (less) search than the original model does. Hence, a lower ratio is better. The vertical lines in the graphs indicate the region in which phase transition occurs, i.e., some of the instances are soluble but some are insoluble. Smith [38] referred such region as the *mushy region*. Note in the graphs for the series $(20, 1)$ that there is only one value of p_2 in the mushy region. Therefore, there is only one vertical line in the graphs for this series.

The peaks for $p_1 = 1, 0.8, 0.6$ are observed at $p_2 = 0.2, 0.25, 0.32$ respectively. They match with their corresponding predictors. Before the mushy

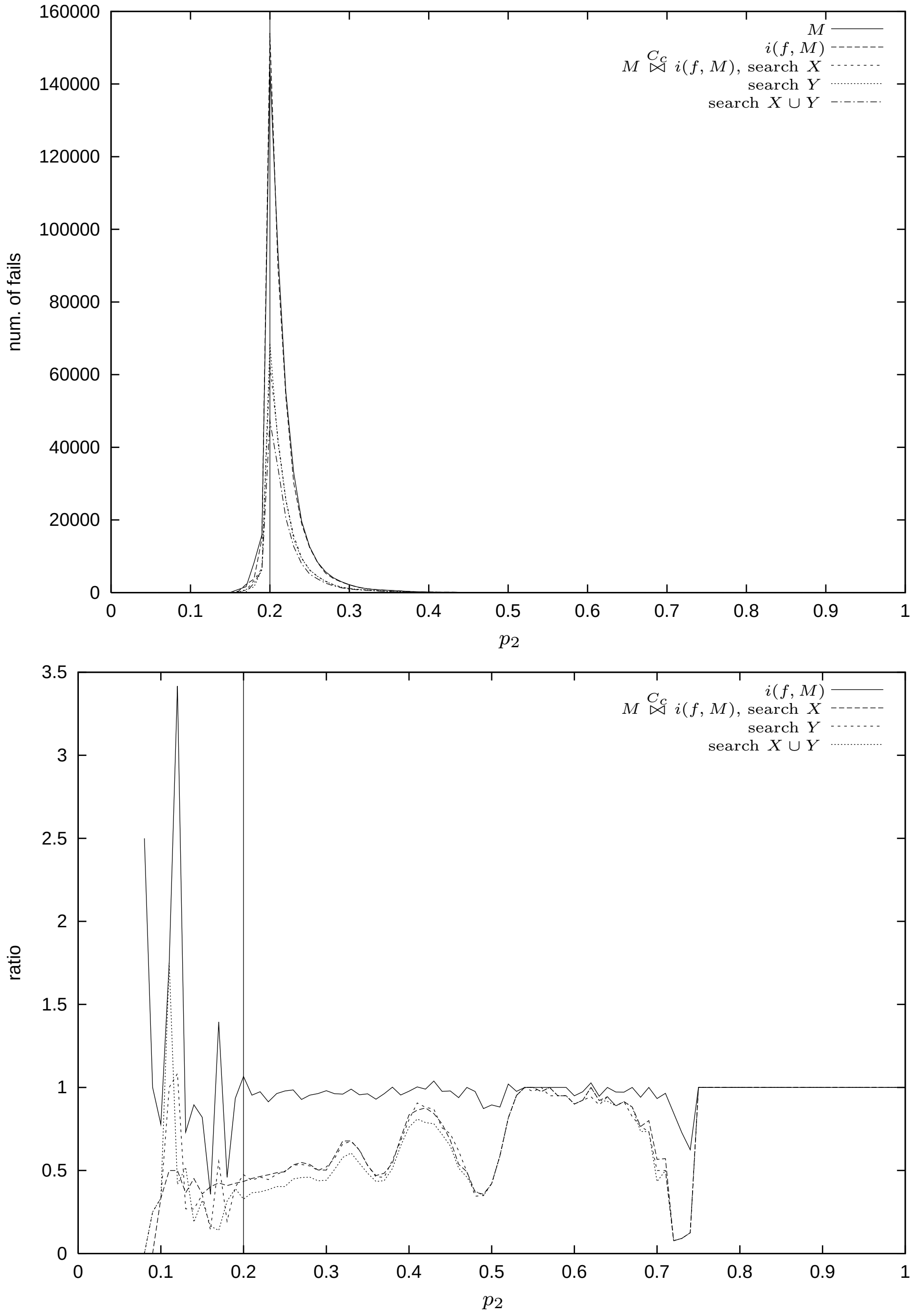


Figure 5.6: Median Number of Fails to Find One Solution or Prove Insolubility for the Series $(20, 1)$, and the Ratios of Fails to the Single Models

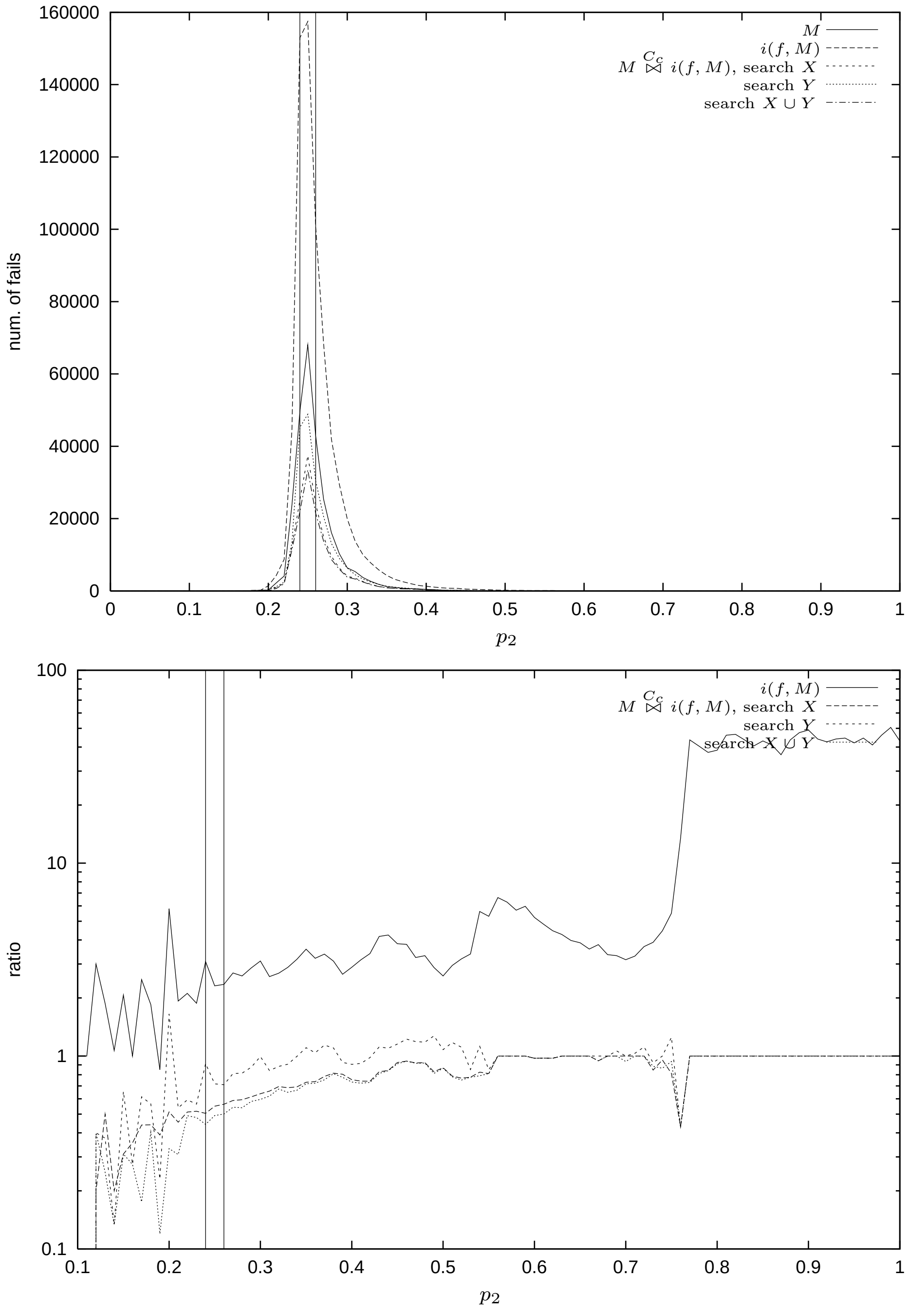


Figure 5.7: Median Number of Fails to Find One Solution or Prove Insolubility for the Series $(20, 0.8)$, and the Ratios of Fails to the Single Models

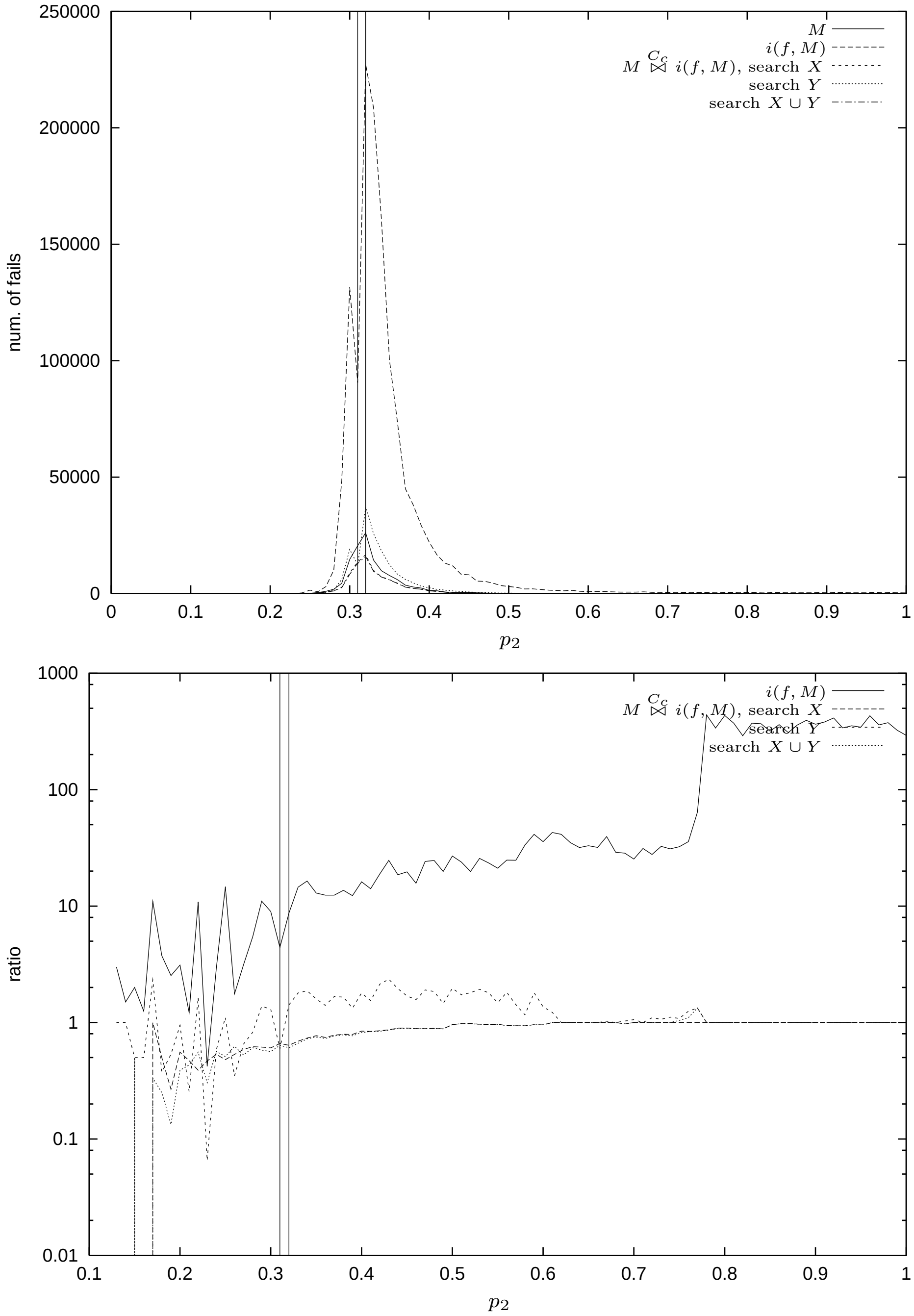


Figure 5.8: Median Number of Fails to Find One Solution or Prove Insolubility for the Series $(20, 0.6)$, and the Ratios of Fails to the Single Models

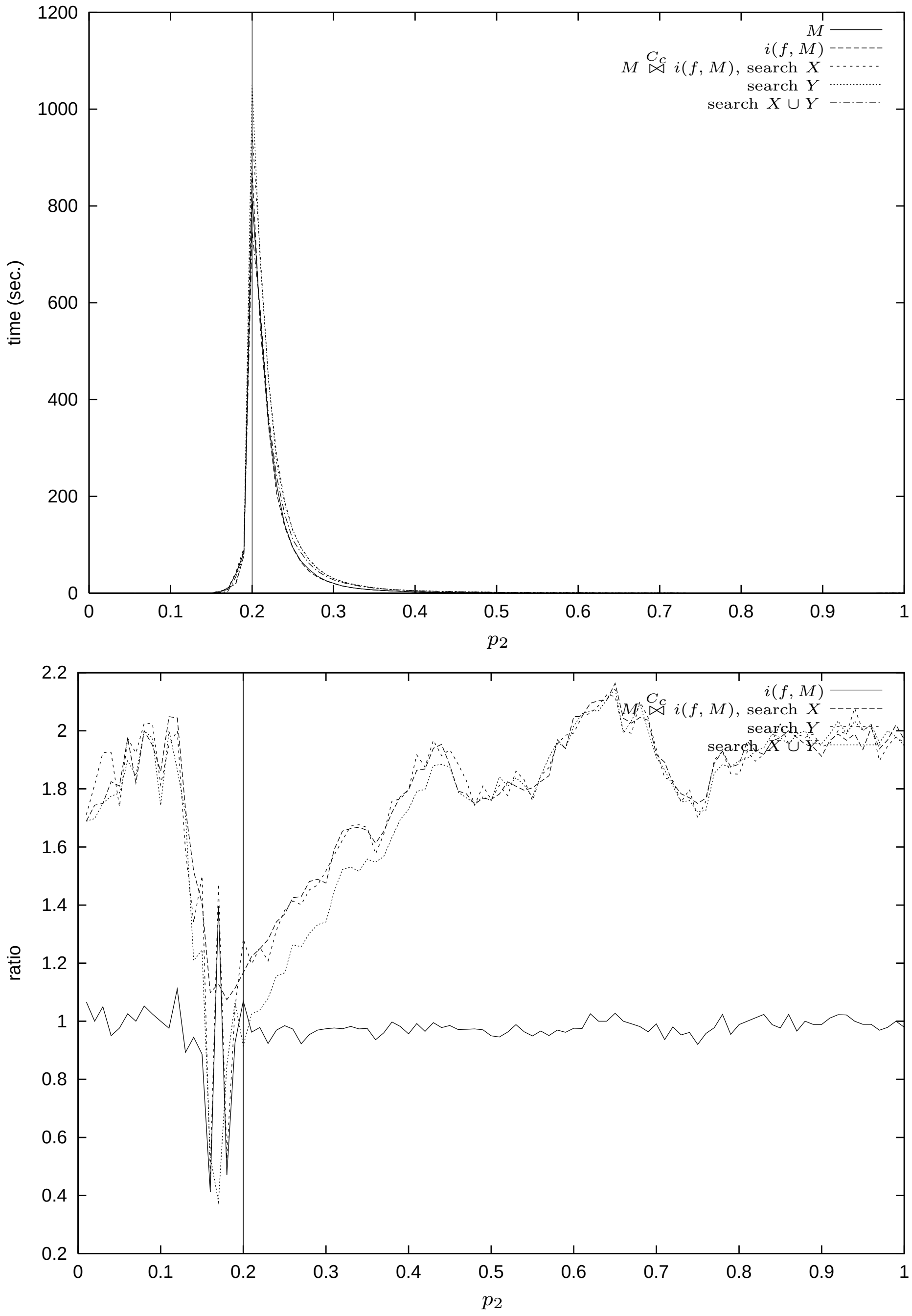


Figure 5.9: Median Running Time to Find One Solution or Prove Insolubility for the Series $(20, 1)$, and the Ratios of Running Time to the Single Models

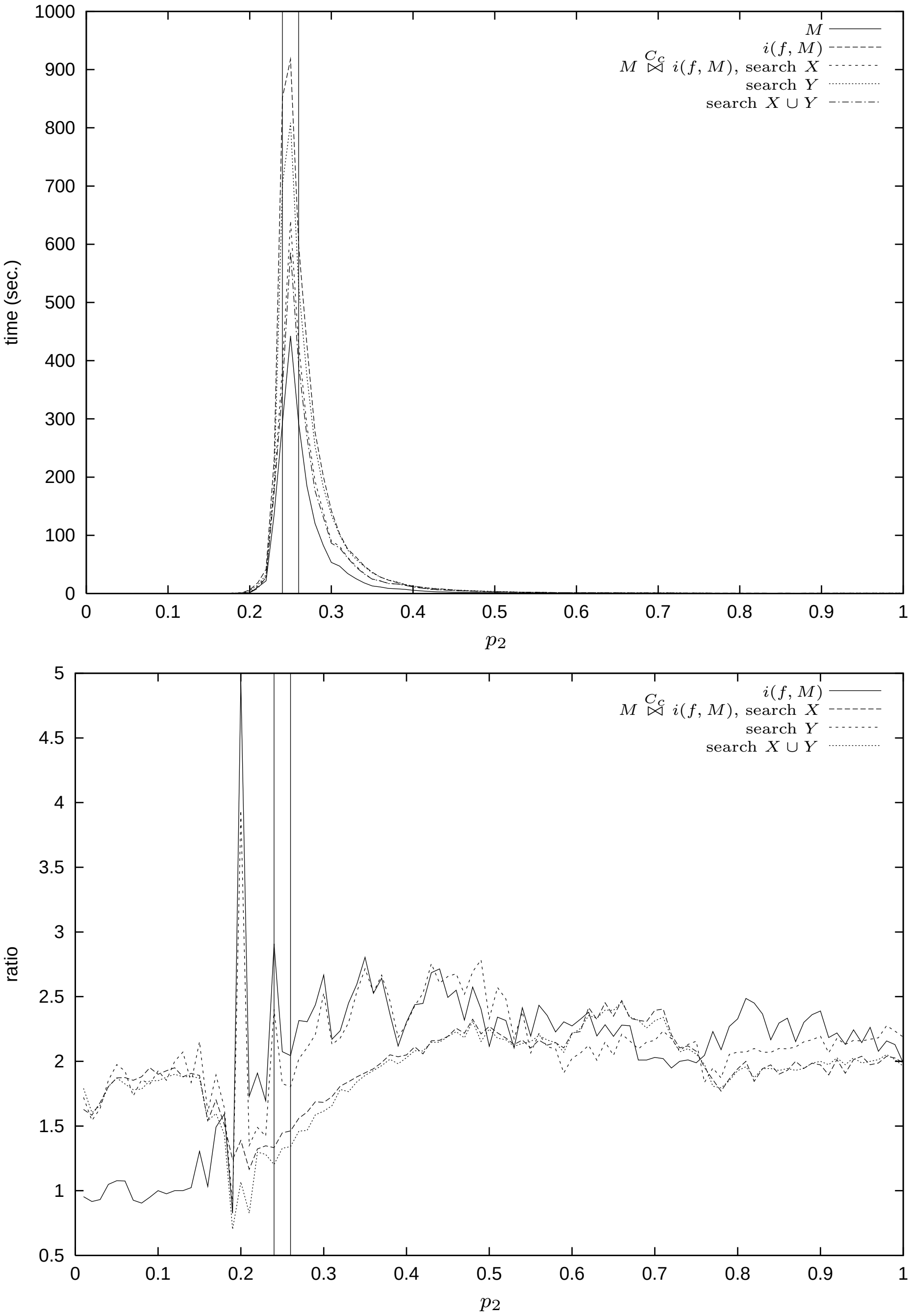


Figure 5.10: Median Running Time to Find One Solution or Prove Insolubility for the Series $(20, 0.8)$, and the Ratios of Running Time to the Single Models

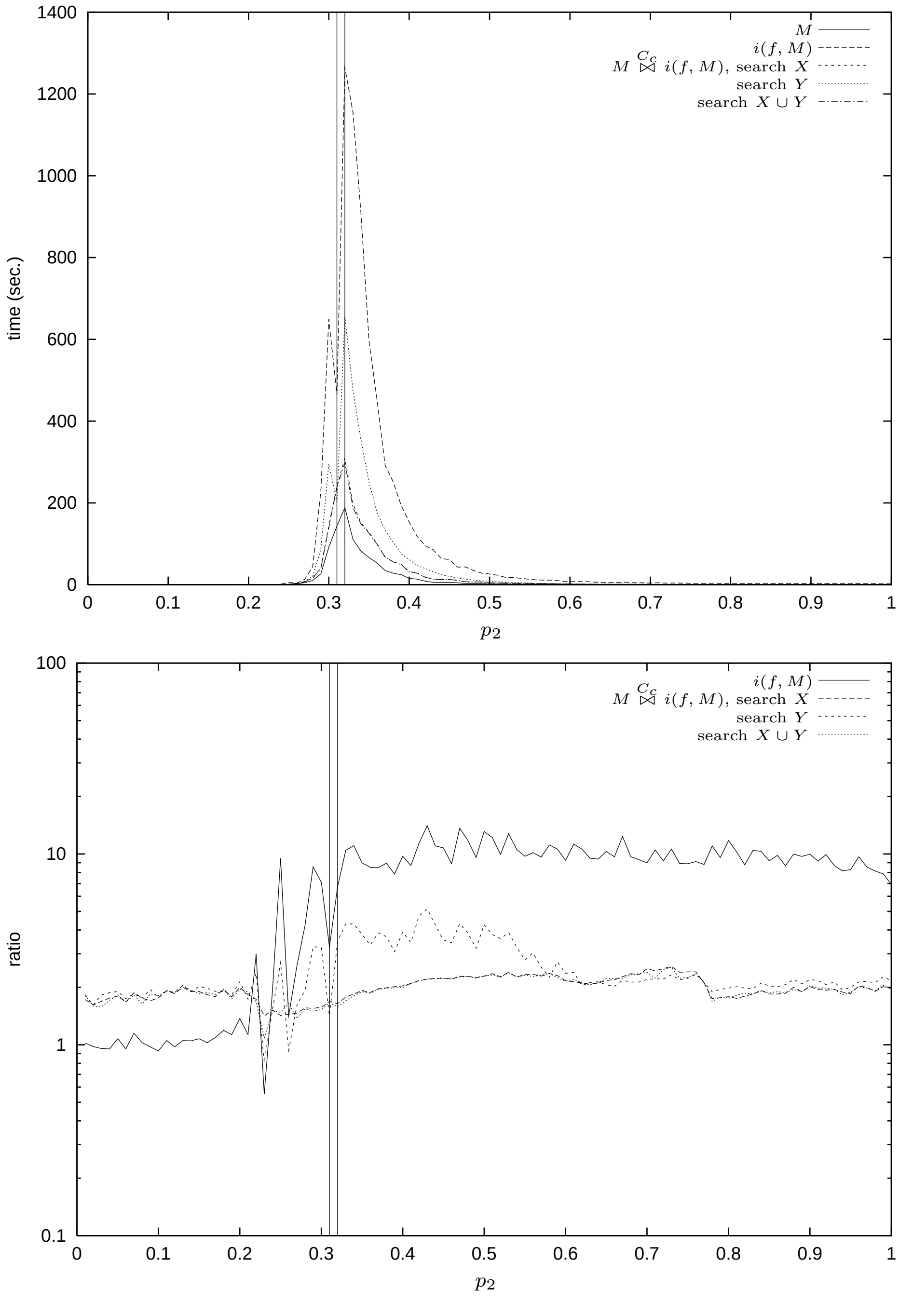


Figure 5.11: Median Running Time to Find One Solution or Prove Insolubility for the Series $(20, 0.6)$, and the Ratios of Running Time to the Single Models

region, the amount of search increases with p_2. After the mushy region, the instances become easier to be proved insoluble and the amount of search drops with increasing p_2. At high constraint tightness, constraint propagation alone is enough to prove the instances insoluble. Therefore, the number of fails is equal to 1.

The channeled models $M \overset{C_c}{\bowtie} i(f, M)$ generally require much less search than the single models M alone, and searching the variables of both models performs better than searching variables in one model only. Therefore, we only focus in the channeled model on searching variables in both models.

The ratios generally increase with p_2 regardless of the phase transitions and gradually converge to one because at high p_2 values, both the original models and channeled models need no search to prove insolubility and the number of fails (and hence the ratio) is one at that case. Note that the ratios of $p_1 = 1$ is generally lower than those of the other p_1 values. Also, at the peak levels, the ratio decreases with increasing p_1. For example, the ratios of $(20, 1, 0.2), (20, 0.8, 0.25)$, and $(20, 0.6, 0.32)$ are 0.33, 0.49, and 0.61 respectively. These indicate that model channeling is useful to CSPs whose variables are highly connected by constraints and can have substantial pruning of the search spaces.

In the figures, we can also see that the induced models alone perform no better than the original model. Only at $p_1 = 1$, the induced models are competitive. At lower p_1 values, they perform much worse than the original models. This is because when p_1 is low in M, the induced model $i(f, M)$ would have a low p_2 value, which means that a value in a variable domain can less likely be pruned. In particular, even at high p_2 levels, we still need a certain amount of search to prove the instances to be insoluble. Figure 5.12 shows an example of such situation. Note that we skip the incompatible assignments of the all-different constraints for clarity. The original model is on the left of Figure 5.12. The constraint between x_1 and x_2 can never be satisfied because

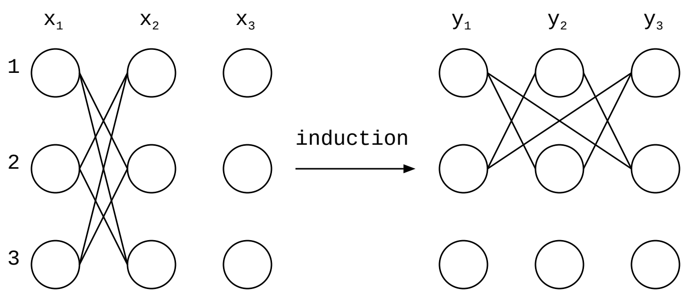


Figure 5.12: An Insoluble CSP and its Induced Model

it contains all the possible incompatible assignments. Besides, we still have the constraints $x_2 \neq x_3$ and $x_1 \neq x_3$ in the model. Before starting the search, constraint propagation alone will prune all the values in the domains of x_1 and x_2. Hence, insolubility is proved without search.

The model on the right of the Figure 5.12 is the induced model after model induction. Clearly, the induced model is arc consistent because the values 1 and 2 of a variable are always compatible with the value 3 in another variable. Hence, constraint propagation will not prune any values from the domains and we need to perform searching in order to prove the induced model to be insoluble. Only when $p_1 = 1$, we can have a chance of detecting arc inconsistencies in the induced model at the beginning of search. Hence, the induced models for $p_1 = 0.6$ and 0.8 alone are not efficient to solve, but they are useful in combining with other models through intersection or channeling to obtain extra redundant information.

Figures 5.9, 5.10, and 5.11 show the timing results of the series $(20, 1)$, $(20, 0.8)$, and $(20, 0.6)$ respectively. Despite the drastic pruning of the search spaces, model channeling does not always lead to a faster running time than using a single model because the channeled models are larger in size and constraint propagation takes longer execution time at each node of the search tree. The amount of timing overhead depends on different p_1 and p_2 levels and the ratios converge to two on high p_2 values. However, in our experiments, there are cases that an instance requires much longer time to run than its corresponding channeled model, and the ratio is more than an order of magnitude. This shows that while a channeled model in general may not be as efficient as a single model, there is potential that in particular circumstances, the channeled model may be substantially more efficient than the single model and can tackle an otherwise computationally infeasible problem. Finally, since the induced models alone for $p_1 < 1$ requires more search than the original models, they have a comparatively longer running time, especially at high p_2 values.

Figures 5.13, 5.14, and 5.15 show the number of fails of different models for finding all solutions for the series $(15, 1)$, $(15, 0.8)$, and $(15, 0.6)$ respectively. We also show the ratio of the amount of search of the induced models or channeled models to that of the corresponding original models in the figures. Contrast to the case of finding one solution or proving no solutions exist, where a peak is found during phase transition, the number of fails decreases smoothly as p_2 increases in the case of finding all solutions. Therefore, phase transition is interesting only when only one solution of the problems is required. The peak predictors for these series also lie between their corresponding mushy regions, showing the accuracy of formula (5.1). The pattern of the curves of the ratios is similar to that of the series $(20, 1)$, $(20, 0.8)$, and $(20, 0.6)$, i.e., the ratio increases with p_2 and decreases with increasing p_1. The timing results in Figures 5.16, 5.17, and 5.18 are also similar to those for finding one solution or proving no solutions exist in Figures 5.9, 5.10, and 5.11. This again shows that a channeled model may not be most competitive in general, but it has the potential of having a substantial pruning of the search space and drastic cut of running time in some circumstances.

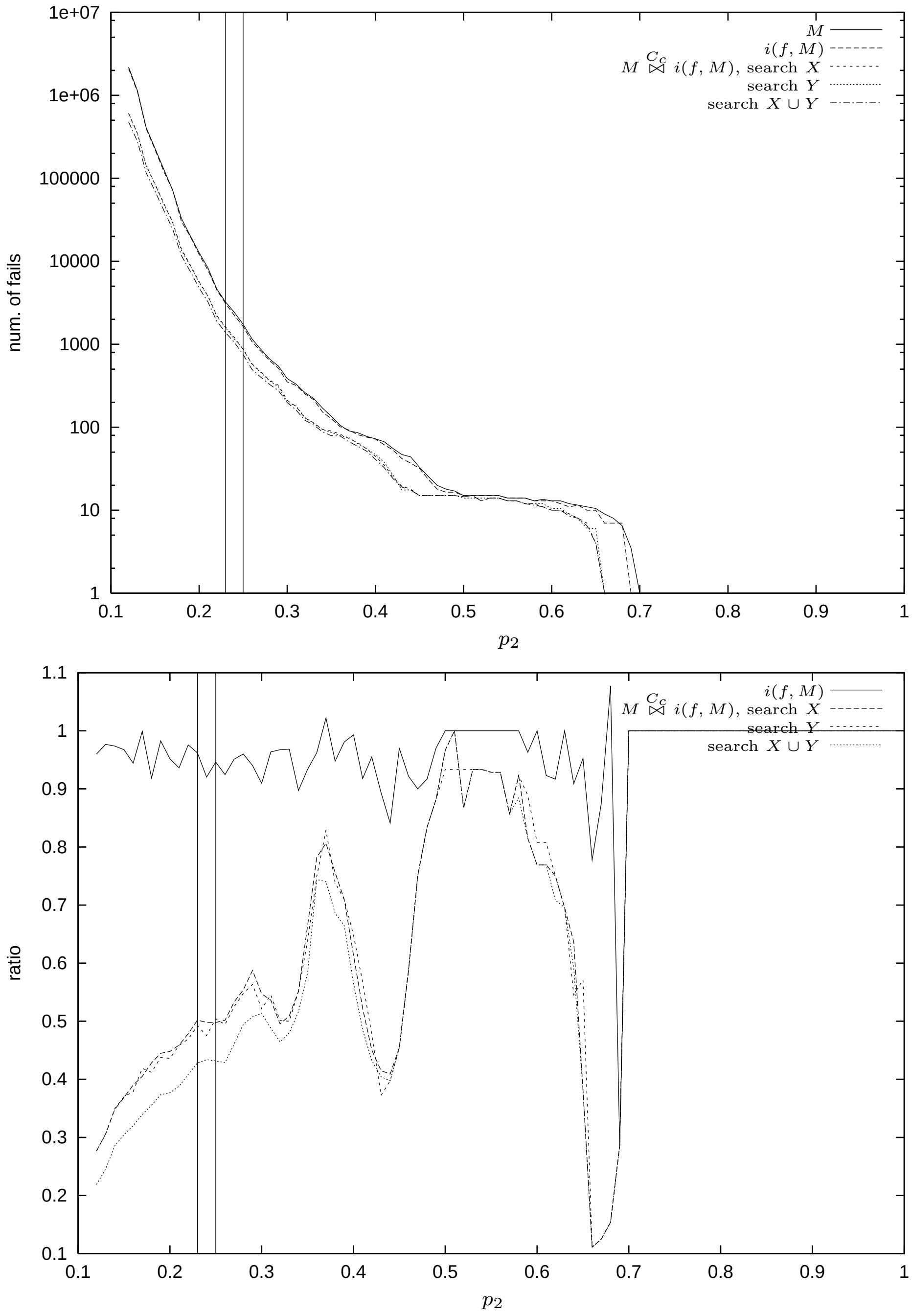


Figure 5.13: Median Number of Fails to Find All Solutions or Prove Insolubility for the Series $(15, 1)$, and the Ratios of Fails to the Single Models

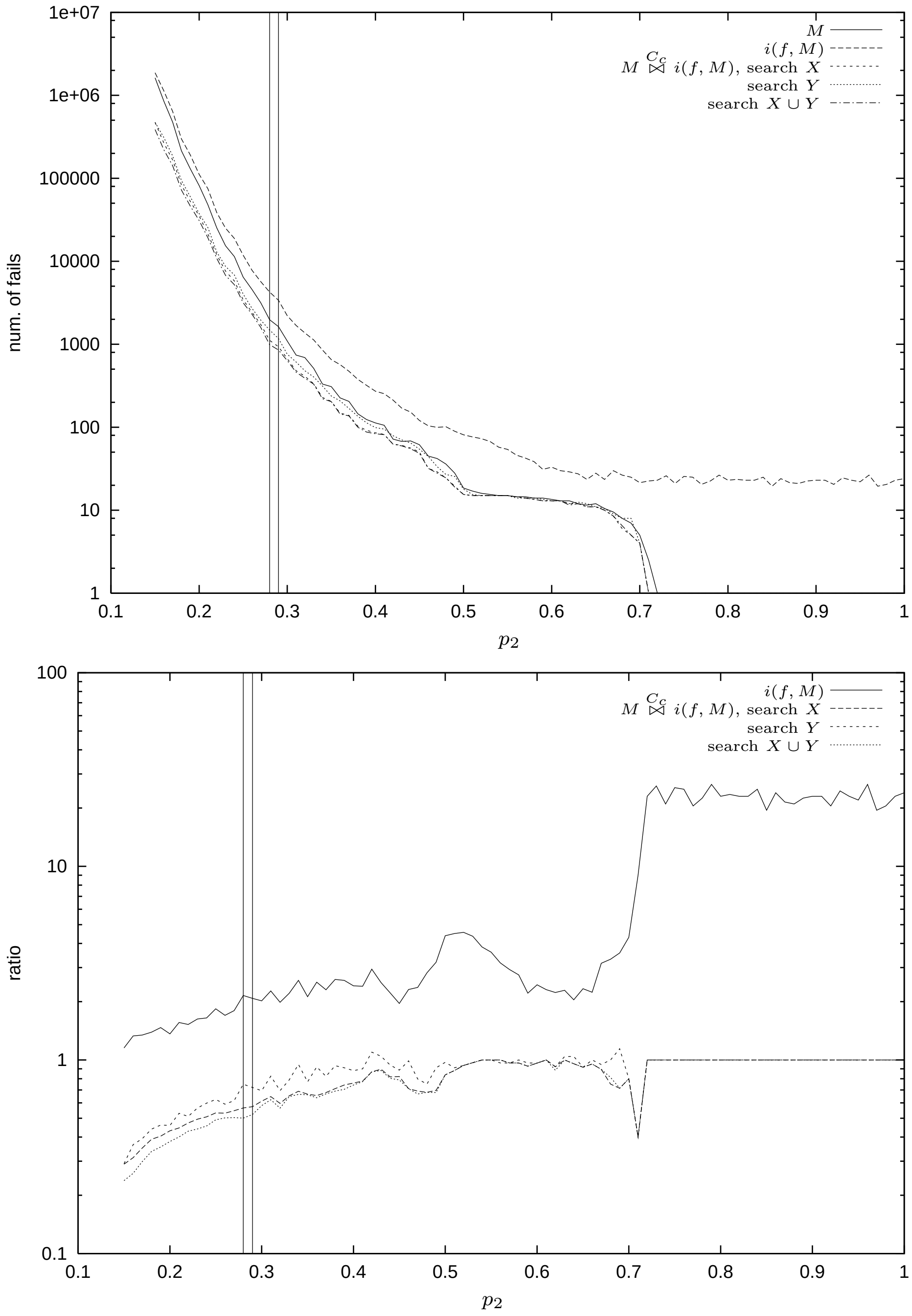


Figure 5.14: Median Number of Fails to Find All Solutions or Prove Insolubility for the Series $(15, 0.8)$, and the Ratios of Fails to the Single Models

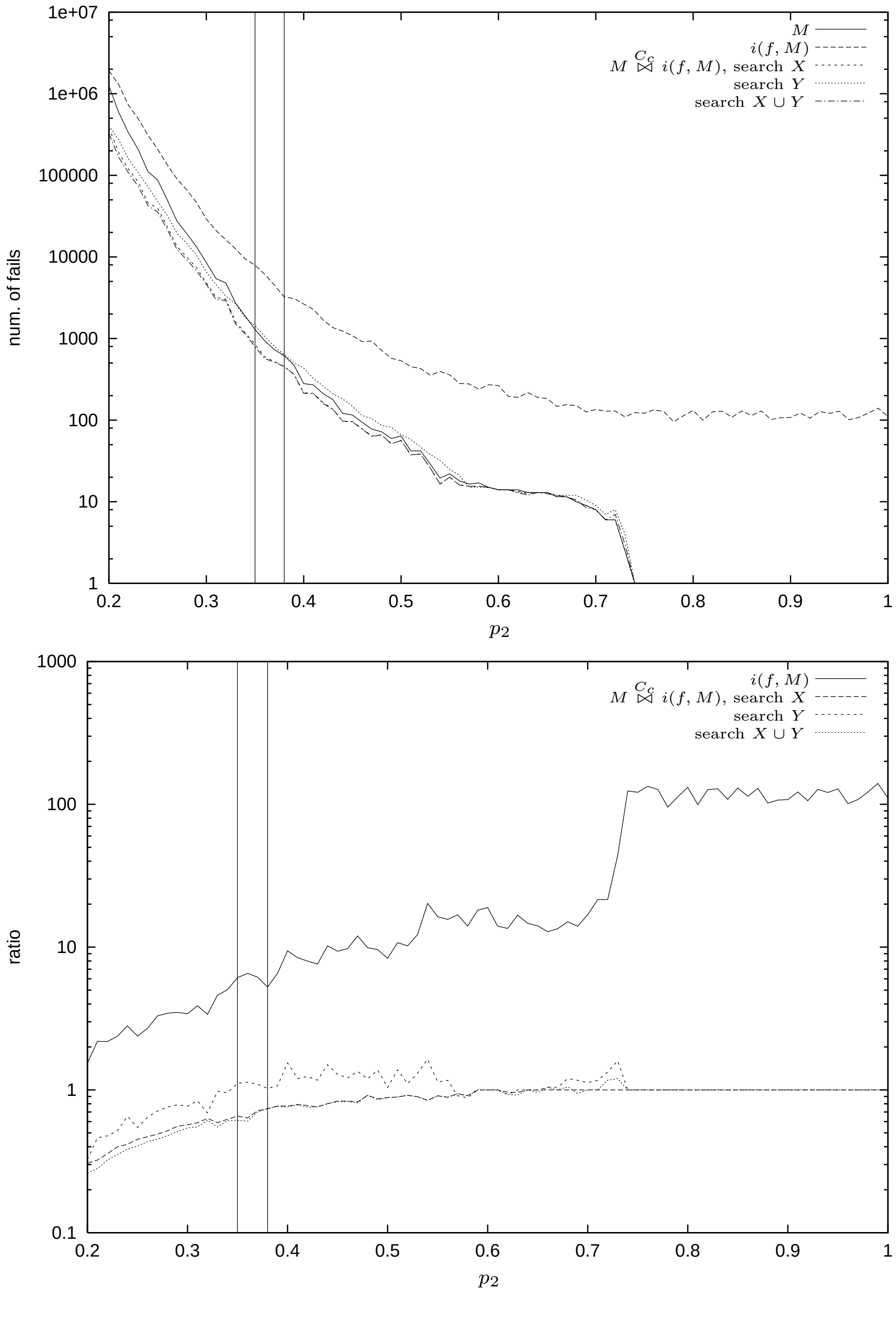


Figure 5.15: Median Number of Fails to Find All Solutions or Prove Insolubility for the Series $(15, 0.6)$, and the Ratios of Fails to the Single Models

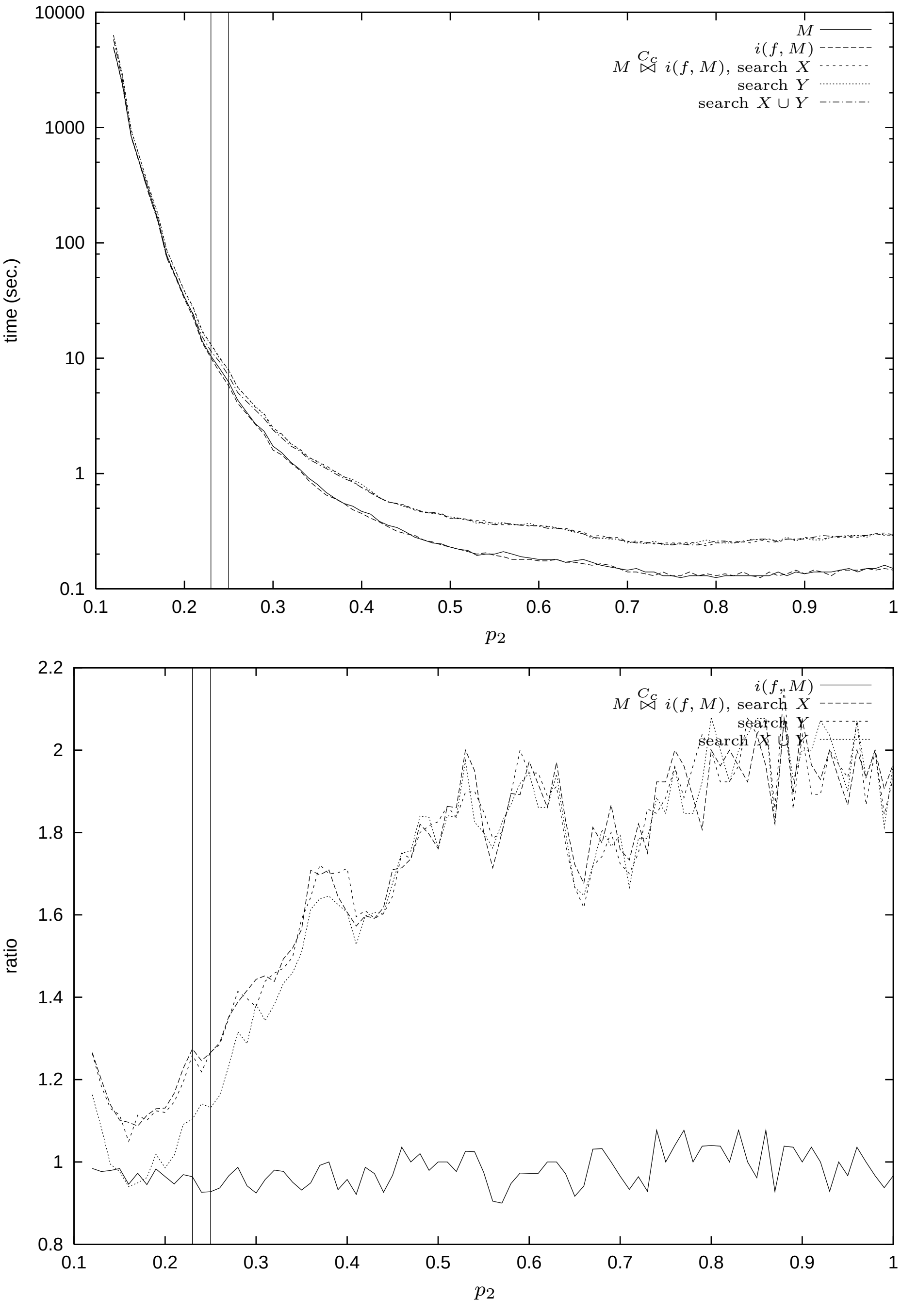


Figure 5.16: Median Running Time to Find All Solutions or Prove Insolubility for the Series $(15, 1)$, and the Ratios of Running Time to the Single Models

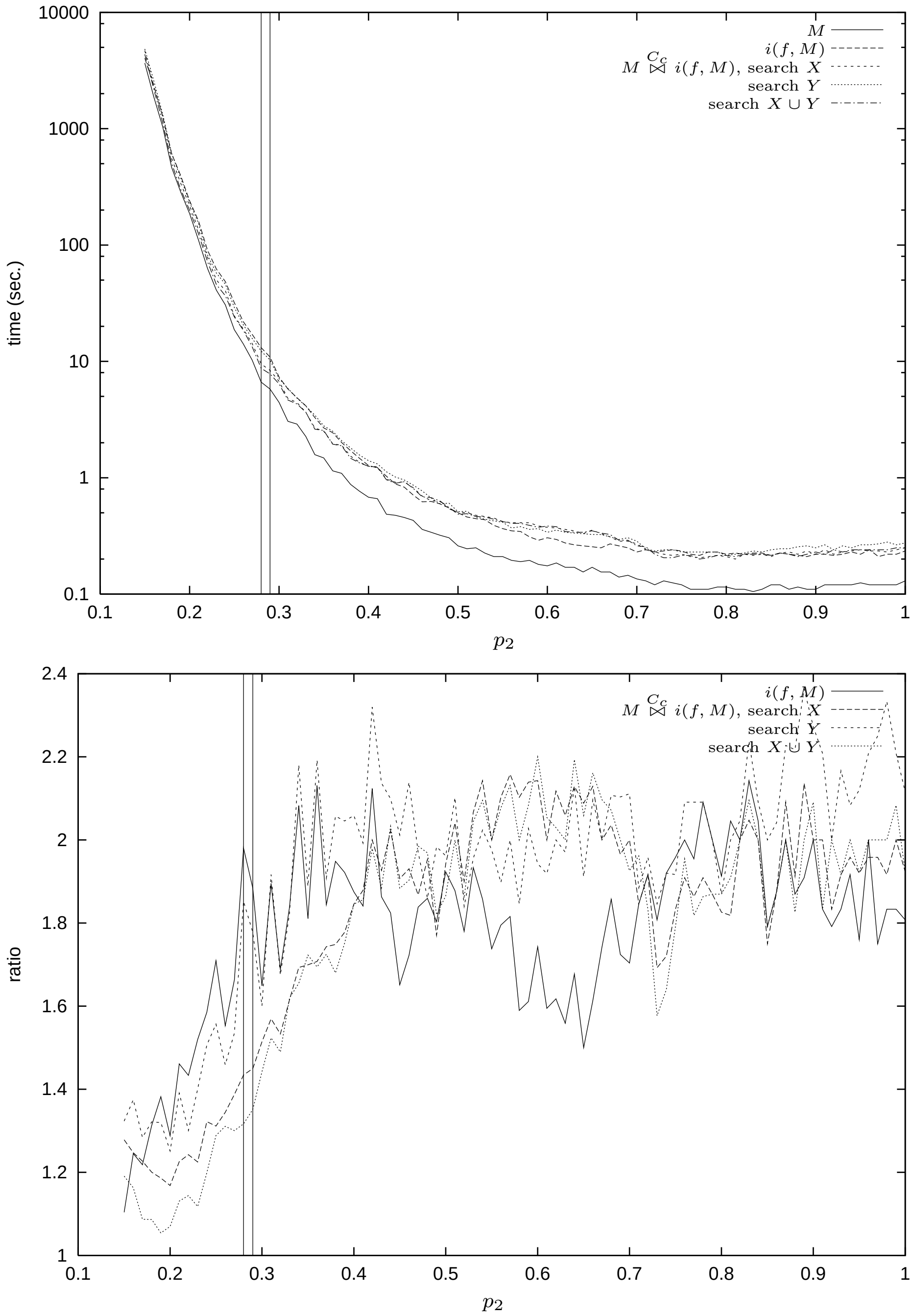


Figure 5.17: Median Running Time to Find All Solutions or Prove Insolubility for the Series $(15, 0.8)$, and the Ratios of Running Time to the Single Models

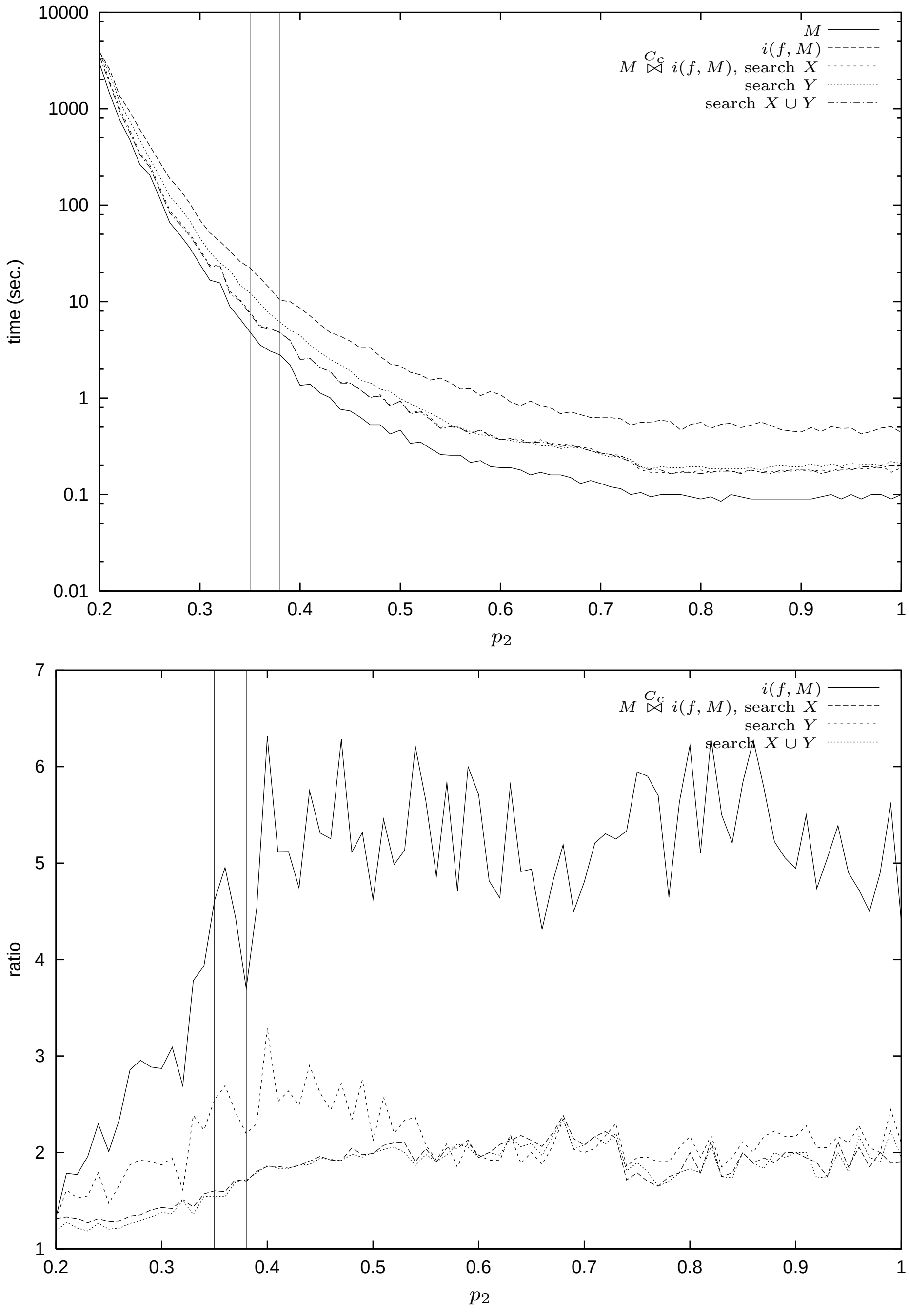


Figure 5.18: Median Running Time to Find All Solutions or Prove Insolubility for the Series $(15, 0.6)$, and the Ratios of Running Time to the Single Models

5.3.3 Golomb Rulers

A Golomb ruler, listed as "prob006" in CSPLib [14], may be defined as a set of m integers $0 = a_1 < \ldots < a_m$ such that the $m(m-1)/2$ differences $a_j - a_i$ for all $1 \leq i < j \leq m$ are all different. Such a ruler is said to have m marks and is of length a_m. The objective is to find optimal (minimal length) rulers, but we can also specify the problem as to obtain a ruler with a given bound on the length.

The Golomb ruler problem can be modeled using m variables $X = \{x_1, \ldots, x_m\}$, representing the marks of the ruler. The domains are $D_X(x_i) = \{0, \ldots, l\}$ for $i \in \{1, \ldots, m\}$, representing the values of the ruler, and l is the upper bound on the length. We note that $l \geq m - 1$ since the number of values of the ruler must be greater than or equal to the number of marks. Using the viewpoint (X, D_X), we can build a model M with the following constraints:

- Monotonicity constraints: $x_1 = 0$ and $x_i < x_{i+1}$ for all $1 \leq i < m$
- Distinct differences constraints: $x_j - x_i \neq x_l - x_k$ for all valid $i < j, k < l$

We can have another viewpoint (Y, D_Y) modeling the Golomb rulers problem. Consider using $l + 1$ variables $Y = \{y_0, \ldots, y_l\}$ representing the values of the ruler. The domains of these variables contain the marks of the ruler, i.e., $1, \ldots, m$. However, since $l \geq m - 1$, there can be some values of the ruler not used by any mark of the ruler. Therefore, we have to introduce artificial marks $m + 1, \ldots, l + 1$ so that every value of the ruler can be assigned marks. Hence, we have the domain $D_Y(y_i) = \{1, \ldots, l + 1\}$ for $i \in \{0, \ldots, l\}$.

A problem of the viewpoint (Y, D_Y) is that we cannot express the constraints of the problem easily. For example, to express the information $a_3 - a_2 \neq a_4 - a_3$, we have to write $(y_i = 2 \wedge y_j = 3) \Rightarrow y_{2j-i} \neq 4$ for all valid i, j pairs. Handcrafting a model using this viewpoint is cumbersome and results in a large amount of weak constraints. Therefore, we shall use this viewpoint for model induction only but not for handcrafting a model.

We need two types of channeling constraints to connect the viewpoints (X, D_X) and (Y, D_Y). The first type is the constraints $x_i = j \Leftrightarrow y_j = i$ for $i \in \{1, \ldots, m\}$ and $j \in \{0, \ldots, l\}$. The second type is the constraints $\forall x \in X \cdot [x \neq i] \Leftrightarrow y_i > m$ for $i \in \{0, \ldots, l\}$ for handling the artificial marks. The second type of channeling constraints clearly do not define a total and injective function. Therefore, we have to modify the original model and introduce artificial variables $x_{m+1}, \ldots, x_{l+1}$ to the viewpoint (X, D_X) in order to be able to apply model induction on them. These variables have the same domain as the original variables, and they should hold values corresponding to the unused values of the ruler. Hence, they must be constrained to be different from the original variables. Now, we have a modified viewpoint (X_p, D_{X_p}) with $X_p = \{x_1, \ldots, x_{l+1}\}$ and $D_{X_p}(x_i) = \{0, \ldots, l\}$ for all $i \in \{1, \ldots, l+1\}$. Besides, we assign an arbitrary order to the artificial variables to maintain a one-one correspondence between the solutions of the original model and the modified model, For example, when $n = 4$ and $l = 6$, we introduce artificial variables x_5, x_6, x_7. Since a solution of a Golomb ruler consists of the assignments of variables $x_1, \ldots, x_4$ only, we impose the arbitrary order $x_5 < x_6 < x_7$ for them so that a solution of the problem corresponds to an unique ordering of the artificial variables. Consider the Golomb ruler $0, 2, 5, 6$. The artificial variables x_5, x_6, x_7 should therefore be assigned $1, 3, 4$ respectively. If we do not apply such an order, any permutations of $1, 3, 4$ can be assigned to the artificial variables to form many solutions which are essentially the same. Below are the two new types of constraints in the modified model M_p constraining the artificial variables:

- $x_i \neq x_j$ for $i \in \{1, \ldots, m\}$ and $j \in \{m+1, \ldots, l+1\}$
- $x_i < x_{i+1}$ for $i \in \{m+1, \ldots, l\}$

By introducing these artificial variables and the corresponding constraints, M_p is now a Permutation CSP. The second type of channeling constraints is not needed. Since we now have a total and injective function defining the channeling constraints connecting (X_p, D_{X_p}) and (Y, D_Y), the induced model $i(f, M_p)$ can be constructed.

When using M_p to solve the Golomb rulers problem, the artificial variables are not needed as the search variables. We only need to search the original variables X to obtain a solution of the problem. However, we still need to search all the variables Y in the induced model $i(f, M_p)$ to obtain a solution.

Table 5.4 shows the comparison results using $(m, l) = (4, 6)$, $(5, 11)$, $(6, 17)$, and $(7, 25)$. The models M_p and M have the same number of fails in all cases, but the execution time for M_p is slightly longer than that of M because there are more variables and constraints in the former models. The channeled models, however, contradict with our conventional wisdom that they should be more efficient in terms of the number of fails, and also behave the same as their corresponding single models in all cases. We cannot come out with a clear explanation of the phenomenon, but this may be due to the use of only part of the variables as search variables in the viewpoint (X_p, D_{X_p}). The induced models cannot have extra constraint propagation due to assignments of the variables in X during the search, and exist solely as overhead in the channeled models. Since the amount of search is not reduced, and the channeled models are larger in size, the execution time for them is longer than that of their corresponding single models.

5.3.4 Circular Golomb Rulers

The circular Golomb rulers problem (or modular Golomb rulers problem) is a variant of the original Golomb rulers problem. A circular Golomb ruler with m marks mod k is a set of m integers $0 = a_1 < \ldots < a_m$ such that the $m(m-1)$ differences $a_i - a_j$ for all $i \neq j$ are all different mod k. It can be thought of as a Golomb ruler with the marks placed on a circle. Note that there are two

Models	Search Variables	First Solution			
		$(4,6)$	$(5,11)$	$(6,17)$	$(7,25)$
M	X	**1** (**0.01**)	**5** (**0.12**)	**21** (**1.42**)	**107** (**15.42**)
M_p	X	**1** (**0.01**)	**5** (0.17)	**21** (1.59)	**107** (16.57)
$i(f, M_p)$	Y	2 (0.03)	167 (1.11)	24860 (594.49)	-
$M_p \overset{C_c}{\bowtie} i(f, M_p)$	X	**1** (0.03)	**5** (0.41)	**21** (3.82)	**107** (37.47)
$M_p \overset{C_c}{\bowtie} i(f, M_p)$	Y	**1** (0.03)	**5** (0.4)	**21** (3.91)	**107** (38.02)
$M_p \overset{C_c}{\bowtie} i(f, M_p)$	$X \cup Y$	**1** (0.04)	**5** (0.4)	**21** (3.79)	**107** (36.88)

Models	Search Variables	All Solutions			
		$(4,6)$	$(5,11)$	$(6,17)$	$(7,25)$
M	X	**5** (**0.01**)	**44** (**0.18**)	**283** (**3.83**)	**2371** (**132.98**)
M_p	X	**5** (**0.01**)	**44** (0.26)	**283** (4.87)	**2371** (150.87)
$i(f, M_p)$	Y	91 (0.16)	15568 (125.08)	-	-
$M_p \overset{C_c}{\bowtie} i(f, M_p)$	X	**5** (0.03)	**44** (0.79)	**283** (15.55)	**2371** (396.21)
$M_p \overset{C_c}{\bowtie} i(f, M_p)$	Y	**5** (0.03)	**44** (0.84)	**283** (16.03)	**2371** (401.02)
$M_p \overset{C_c}{\bowtie} i(f, M_p)$	$X \cup Y$	**5** (0.04)	**44** (0.77)	**283** (15.49)	**2371** (395.13)

Table 5.4: Comparison Results using the Golomb Rulers Problem

differences between two marks i and j. One is the difference $a_i - a_j \bmod k$ and the other is $a_j - a_i \bmod k$. They are equal to the distances from one mark to another along the two directions on the circle. Hence, the sum of the differences is equal to the circumference of the circle.

There are several good constructions known for circular Golomb rulers. Singer [35] shows for every prime power (power of a prime number) p there exists a modular Golomb ruler with $p + 1$ marks mod $p^2 + p + 1$. These rulers are known as perfect difference sets, i.e., the $m(m-1)$ differences ranges from 1 to $m(m-1)$ inclusive. We are interested in tackling circular Golomb rulers with m marks mod k where $k = (m-1)^2 + (m-1) + 1 = m^2 - m + 1$ because they can be modeled as Permutation CSPs also. Hence, when $m - 1$ is a prime power, there must be solutions to the problem.

We model the problem in a different way as the original Golomb rulers problem. Instead of using variables to represent the marks, we use the variables $Z = \{z_{ij} | i, j \in \{1, \ldots, m\} \wedge i \neq j\}$ to represent the difference $a_j - a_i \mod k$ of the marks. The domains of the variables $D_Z(z_{ij}) = \{1, \ldots, m(m-1)\}$ for all i, j represent the values that the differences can take.

By using the viewpoint (Z, D_Z), we can construct a model M using four types of constraints. The reflexive constraints enforce the modulo relationship between z_{ij} and z_{ji} for $i < j$. The transitive constraints define the intermediate roles of the variables z_{ij} for $1 < i < j \leq m$ between z_{1i} and z_{1j}. The all-different constraints ensure that the differences among the marks in the ruler are all distinct. The symmetry breaking constraints remove symmetries due to rotation and reflection of the circle. Rotation is removed by arbitrarily setting $a_2 = 1$ and hence $z_{12} = 1$, while reflection is removed by imposing an order between z_{23} and z_{m1}. The constraints can be summarized as follows:

- Reflexive constraints: $z_{ij} + z_{ji} = k$ for $1 \leq i < j \leq m$
- Transitive constraints: $z_{1i} + z_{ij} = z_{1j}$ for $1 < i < j \leq m$

- All-different constraints: $z_{ij} \neq z_{kl}$ for $i, j, k, l \in \{1, \ldots, m\}$ and $i \neq k \vee j \neq l$

- Symmetry breaking constraints $z_{12} = 1$ and $z_{23} < z_{m1}$

Again, not all variables in Z have to be search variables. Since the variables $Z_s = \{z_{1,2}, \ldots, z_{1,m}\}$ define all the other variables in the viewpoint, we only need to search these variables to obtain solutions of the problem.

Since the previous model is a Permutation CSP, we can always have another viewpoint with interchanging roles of variables and values. We can use variables $V = \{v_1, \ldots, v_{m(m-1)}\}$ to represent the values of the differences of the marks. The domains of these variables $D_V(v_i) = \{(i, j)|i, j \in \{1, \ldots, m\} \wedge i \neq j\}$ represent the pair of different marks of the ruler corresponding to the distance from mark i to j. The viewpoints (Z, D_Z) and (V, D_V) can be connected using the channeling constraints $z_{ij} = k \Leftrightarrow v_k = (i, j)$ or $f(\langle z_{ij}, k\rangle) = \langle v_k, (i, j)\rangle$ for all $i, j \in \{1, \ldots, m\}, k \in \{1, \ldots, m(m-1)\}$ and $i \neq j$.

Since 4 and 5 are prime power $(2 \times 2 = 4)$ and prime respectively, there are circular Golomb rulers for $m = 5$ and $m = 6$. Table 5.5 shows the comparison results for finding first solution and all solutions of these circular Golomb rulers. We choose these relatively small instances because the memory requirement for holding the induced model is large. There are 197 and 446 constraints in the original models respectively, while there are 1349 and 4524 constraints in their corresponding induced models. This means model induction transforms the original models to the induced models using more but weaker constraints, and thus incurs a large overhead to tackle the problem. Hence, we have to use another Sun Ultra 5/400 workstation with 512M memory to obtain the experimental results.

The results obtained are similar to the Langford's problem. The channeled models are always more efficient than the single models in terms of the number

Models	Search Variables	First Solution $m=5$, $k=21$	First Solution $m=6$, $k=31$	All Solutions $m=5$, $k=21$	All Solutions $m=6$, $k=31$
M	Z_s	12 (**1.1**)	5 (**10.55**)	23 (**1.29**)	115 (**18.7**)
$i(f,M)$	V	-	-	-	-
$M \overset{C_c}{\bowtie} i(f,M)$	Z_s	11 (4.7)	**3** (26.18)	19 (6.37)	**70** (95.87)
$M \overset{C_c}{\bowtie} i(f,M)$	V	**10** (5.63)	**3** (26.27)	**13** (6.93)	101 (164.21)
$M \overset{C_c}{\bowtie} i(f,M)$	$Z_s \cup V$	**10** (5.52)	**3** (25.82)	**13** (6.76)	101 (159.05)

Table 5.5: Comparison Results Using the Circular Golomb rulers problem

of fails. However, since the memory overhead of the channeled models is large, the best timing always belongs to the original single models. The induced model and the channeled model for $m = 7$, although unsatisfiable, require so much memory that we fail to obtain experimental results for them within reasonable time.

5.3.5 All-Interval Series Problem

The all-interval series problem, listed as "prob007" in CSPLib [14], is another integer sequence problem that can be modeled as a Permutation CSP. The goal is to find a permutation of $0, \ldots, n-1$ such that the absolute differences of any consecutive pair of numbers are all distinct. Simonis and Beldiceanu [34] note that a first solution can be found without search using global constraints with the regular sequence:

$$0, n-1, 1, n-2, 2, n-3, \ldots$$

Therefore, our interest remains in finding all solutions to the problem.

To model the problem, we use the variables $X = \{x_0, \ldots, x_{n-1}\}$ to represent the positions in the sequence. The domains of the variables are the values that occur in the sequence. Hence, we have the viewpoint (X, D_X), where $D_X(x_i) = \{0, \ldots, n-1\}$ for $i \in \{0, \ldots, n-1\}$.

Using this viewpoint, we can formulate the problem into a model M using two types of constraints. The all-different constraints ensure that all positions are occupied by different numbers. The difference constraints ensure that the differences between any pair of consecutive numbers are all different.

- all-different constraints: $x_i \neq x_j$ for all $0 \leq i < j < n$
- difference constraints: $|x_i - x_{i+1}| \neq |x_j - x_{j+1}|$ for all $0 \leq i < j \leq n-2$

The reverse of any solution sequence is also a solution of the problem. We can remove this symmetry by imposing a third type of symmetry breaking constraint $x_0 < x_{n-1}$ to obtain the asymmetric model M_{as}.

Since M and M_{as} are Permutation CSPs, we can again interchange the roles of the variables and values to obtain another viewpoint of the problem. We use the variables $Y = \{y_0, \ldots, y_{n-1}\}$ to represent the numbers in the sequence. The domains of these variables are the position of the numbers in the sequence. Therefore, we have another viewpoint (Y, D_Y), where $D_Y(y_i) = \{0, \ldots, n-1\}$ for $i \in \{0, \ldots, n-1\}$.

The channeling constraints connecting (X, D_X) and (Y, D_Y) are $x_i = j \Leftrightarrow y_j = i$, or $f(\langle x_i, j\rangle) = \langle y_j, i\rangle$, for $i, j \in \{0, \ldots, n-1\}$. With M and M_{as}, we can construct the induced models $i(f, M)$ and $i(f, M_{as})$, and channeled models $M \overset{C_c}{\bowtie} i(f, M)$ and $M_{as} \overset{C_c}{\bowtie} i(f, M_{as})$. Table 5.6 shows the comparison results for finding all solutions of the all-interval series problem of size $n = 5, \ldots, 9$.

In this problem, model channeling is not too successful in reducing the amount of search. The reduction in number of fails is only 0% to 5%, but the time required for the channeled models is about double of that for the single models. This is because the size of the channeled models is about twice of that of the single models. This phenomenon is related to the difficulty of the problem. Table 5.7 shows the number of solutions of different problem sizes. We can see that there are numerous solutions to an instance in a relatively small search space. That makes the failure of the search occur mainly in

Models	Search Variables	*n*				
		5	6	7	8	9
M	X	52 (0.05)	178 (0.19)	702 (1.1)	2938 (6.94)	12846 (40.44)
$i(f, M)$	Y	58 (0.04)	218 (0.25)	968 (1.5)	4368 (10.96)	21368 (81.8)
M_{as}	X	**29** (0.03)	**104** (**0.12**)	445 (**0.71**)	1979 (**4.59**)	9147 (**27.84**)
$i(f, M_{as})$	Y	45 (0.04)	167 (0.17)	696 (1.13)	3070 (8.19)	14543 (57.57)
$M \overset{C_c}{\bowtie} i(f, M)$	X	52 (0.06)	178 (0.36)	696 (2.05)	2888 (14.15)	12730 (93.07)
$M \overset{C_c}{\bowtie} i(f, M)$	Y	54 (0.06)	194 (0.37)	796 (2.21)	3545 (16.95)	16899 (117.4)
$M \overset{C_c}{\bowtie} i(f, M)$	$X \cup Y$	52 (0.06)	178 (0.34)	696 (2.11)	2888 (14.25)	12730 (92.4)
$M_{as} \overset{C_c}{\bowtie} i(f, M_{as})$	X	**29** (**0.02**)	**104** (0.22)	425 (1.24)	1882 (8.41)	8736 (58.35)
$M_{as} \overset{C_c}{\bowtie} i(f, M_{as})$	Y	**29** (0.04)	**104** (0.23)	**420** (1.27)	**1857** (9.18)	8922 (64.28)
$M_{as} \overset{C_c}{\bowtie} i(f, M_{as})$	$X \cup Y$	**29** (0.04)	**104** (0.22)	428 (1.22)	1874 (8.45)	**8712** (57.68)

Table 5.6: Comparison Results for Finding All Solutions Using the All-interval Series Problem and the Number of Solutions of the Problem

n	With Symmetries	Without Symmetries
5	8	4
6	24	12
7	32	16
8	40	20
9	120	60

Table 5.7: Number of Solutions of Different Instances of the All-interval Series Problem

the bottom part of the search tree. Hence, we can hardly obtain any earlier failures even when model channeling allows more constraint propagation. We conjecture that model channeling is beneficial to problems in which solutions are rare or not evenly distributed in the search space, and a lot of search effort is required to obtain a solution.

Symmetry breaking is more successful in reducing the amount of search, and the M_{as}-based models require the shortest time in all cases. Note that the amount of search reduction of the channeled models with symmetry breaking is generally higher than that without symmetry breaking. This conforms with our conjecture on the difficulty of the problem because the number of solutions is reduced by half with symmetry breaking. Earlier failures due to constraint propagation of the channeled model can occur more frequently.

Chapter 6

Concluding Remarks

We conclude the thesis in this chapter by giving our contributions and possible directions of future works.

6.1 Contributions

The contribution of our work can be summarized as follows. First, we provide formal definitions on viewpoints and models of CSPs, and the notion of model redundancy. Second, based on these definitions, we introduce model induction, a systematic way of generating alternate model in a different viewpoint from an existing model. We give its syntactic construction rule, detailed examples, and also examine its properties. It allows an automatic generation of mutually redundant models. Third, we propose two operators for combining models, namely model intersection and model channeling. Model intersection allows combining two models of the same problem in the same viewpoint, while model channeling allows combining two models in different viewpoints with the use of channeling constraints. The latter generalizes the idea of redundant modeling by combining the constraints and defining a relationship between the two viewpoints of the constituent models with the use of channeling constraints. Fourth, we identify three new forms of redundancy by combining models using model intersection, channeling, and induction. Hence, we can utilize redundant information in enhancing constraint propagation. Our benchmark results, especially in the Langford's problem and random Permutation CSPs, confirm that the proposed combined models are robust and efficient. Fifth, handcrafting CSP model is an unamiable and costly task performed daily by human modelers. The time for model induction to generate an induced model, say, for the Langford's problem, takes about 2 to 17 seconds depending on the instances. Therefore, they should find model induction a useful tool. Since model redundancy is a relatively new concept, our formal study helps advancing our understanding of model redundancy, and when redundant information can arise and is useful. Our work is also a means to open up new possibilities to study, understand, and apply model redundancy in constraint satisfaction.

6.2 Future Work

Our work proposes a systematic study of model operators for transforming and combining CSP models. Our study arouses interest in this important new direction of research. There is plenty of scope for future work. First, it would be interesting to refine the definition of model induction. In particular, although our definition of model induction is applicable to general CSPs, our empirical results are developed for only Permutation CSPs. It will be interesting to check if the same techniques can be applied/generalized to other not necessarily Permutation CSPs to obtain useful redundant information. For example, the requirement of a total and injective function defined by channeling constraints is quite restrictive. Further study can be conducted, perhaps, to refine requirement on the type of channeling constraints that model induction can apply.

Second, model induction is defined in terms of extensional representation of constraints. There is no reason why we have to solve the resultant CSPs also in the extensional form because propagating an extensional constraint in a model is generally less efficient its intensional (symbolic) counterpart in many constraint programming systems. It would be worthwhile to study how the intensional representation of a constraint can be learned from its extensional counterpart. By writing the constraints in an induced model in symbolic forms, the induced model and hence the channeled model can be solved in a more efficient way.

Third, Permutation CSPs form an important and interesting class of CSPs. In our experiments, we use binary disequality constraints to ensure that the variables takes all-different values. It would be interesting to study the effect of our three new forms of redundancy using GAC all-different constraints. Besides, Smith [37] and Walsh [39] shows that the channeling constraints alone is sufficient to ensure that all variables takes different values in a Permutation CSP, The pruning behaviour of the channeling constraints is even better than that of the set of disequality constraints but worse than that of a GAC all-different constraint. It would be worth investigating how the pruning behaviour of different channeling constraints affects the efficiency of combined models.

Fourth, we show that for binary Permutation CSPs, applying model induction twice is idempotent. This is not the case in general for non-binary Permutation CSPs. Consider an incompatible assignment $\{\langle x_1, 1\rangle, \langle x_2, 1\rangle, \langle x_3, 2\rangle\}$. This incompatible assignment cannot be transformed upon the first induction, because both x_1 and x_2 takes the value 1. Therefore, we "lost" this incompatible assignment upon the second induction. We conjecture that by applying model induction twice to a non-binary Permutation CSP, the resultant model would become final, i.e., the resultant model would be the same as that by applying induction for any even number of times. It would be worthwhile to study whether our conjecture is true.

Bibliography

[1] C. Bessiere. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65(1):179–190, 1994.

[2] C. Bessiere, E. C. Freuder, and J.-C. Regin. Using constraint meta-knowledge to reduce arc consistency computation. *Artificial Intelligence*, 107(1):125–148, 1999.

[3] C. A. Brown and P. W. Purdom. How to search efficiently. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, pages 588–594, 1981.

[4] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 331–337, 1991.

[5] B. M. W. Cheng, , J. H. M. Lee, and J. C. K. Wu. A constraint-based nurse rostering using a redundant modeling approach. In *Proceedings of the Eighth IEEE International Conference on Tools with Artificial Intelligence*, pages 140–148, 1996.

[6] B. M. W. Cheng, , J. H. M. Lee, and J. C. K. Wu. Speeding up constraint propagation by redundant modeling. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming*, pages 91–103, 1996.

[7] B. M. W. Cheng, , J. H. M. Lee, and J. C. K. Wu. A nurse rostering system using constraint programming and redundant modeling. *IEEE Transactions in Information Technology in Biomedicine*, 1(1):44–54, 1997.

[8] B. M. W. Cheng, K. 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.

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

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

[11] E. C. Freuder. In pursuit of the holy grail. *Constraints*, 2(1):57–61, 1997.

[12] J. Gaschnig. A general backtracking algorithm that eliminates most redundant tests. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, page 457, 1977.

[13] P. A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 31–35, 1992.

[14] I. P. Gent and T. Walsh. CSPLib: A benchmark library for constraints. In *Proceedings of Principles and Practice of Constraint Programming (CP)*, pages 480–481, 1999. Available at http://www-users.cs.york.ac.uk/~tw/csplib/.

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

[16] S. A. Grant and B. M. Smith. The phase transition behavior of maintaining arc consistency. In *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI-96)*, pages 175–179, 1996.

[17] R. M. Haralick and G. L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[18] ILOG. *ILOG Solver 4.4 Reference Manual*, 1999.

[19] J. Jourdan. *Concurrent constraint multiple models in CLP and CC languages: Toward a programming methodology by modelling.* PhD thesis, Denis Diderot University, Paris VII, 1995.

[20] J. Jourdan. Concurrent constraint multiple models in CLP and CC languages: Toward a programming methodology by modelling. In *Proceedings of the INFORMS Conference*, 1995.

[21] Y. C. Law and J. H. M. Lee. Algebraic properies of CSP model operators. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, 2002.

[22] Y. C. Law and J. H. M. Lee. Model induction: a new source of CSP model redundancy. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 54–60, 2002.

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

[24] K. Marriott and P. J. Stuckey. *Programming with Constraints.* The MIT Press, 1998.

[25] K. Marriott and P. J. Stuckey. *Programming with Constraints*, chapter 8, pages 266–271. The MIT Press, 1998.

[26] R. Mohr and T. C. Henderson. Arc and path consistency revised. *Artificial Intelligence*, 28(2):225–233, 1986.

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

[28] B. A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989.

[29] M. Perlin. Arc consistency for factorable relations. *Artificial Intelligence*, 53(2–3):329–342, 1992.

[30] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, New York, 1978.

[31] F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 550–556, 1990.

[32] D. Sabin and E. C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming, PPCP'94*, volume 874, pages 10–20, 1994.

[33] C. Schulte and P. J. Stuckey. When do bounds and domain propagation lead to the same search space. In *Proceedings of the Third International Conference on Principles and Practice of Declarative Programming*, pages 115–126, 2001.

[34] H. Simonis and N. Beldiceanu. A note on CSPLib prob007. Technical report, COSYTECH, 1998.

[35] J. Singer. A theorem in finite projective geometry and some applications to number theory. *Transactions American Mathematical Society*, 43:377–385, 1938.

[36] B. M. Smith. Modelling a permutation problem. Research Report 2000.18, School of Computer Studies, University of Leeds, 2000.

[37] B. M. Smith. Dual models in permutation problems. In *Proceedings of Principles and Practice of Constraint Programming*, pages 615–619, 2001.

[38] B. M. Smith and M. E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):155–181, 1996.

[39] T. Walsh. Permutation problems and channelling constraints. In *Proceedings of Logic for Programming, Artificial Intelligence and Reasoning*, pages 377–391, 2001.

[40] R. Weigel and C. Bliek. On reformulation of constraint satisfaction problems. In *Proceedings of 13th European Conference on Artificial Intelligence*, pages 254–258, 1998.