Redundant Modeling in Permutation Weighted Constraint Satisfaction Problems

Y.C. Law J.H.M. Lee M.H.C. Woo
Department of Computer Science and Engineering,
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
{yclaw,jlee,hcwoo}@cse.cuhk.edu.hk

Abstract

In classical constraint satisfaction, redundant modeling has been shown effective in increasing constraint propagation and reducing search space for many problem instances. In this paper, we investigate, for the first time, how to benefit the same from redundant modeling in *weighted constraint satisfaction problems* (WCSPs), a common soft constraint framework for modeling optimization and over-constrained problems. Our work focuses on a popular and special class of problems, namely, the *permutation problems*. First, we show how to automatically generate a redundant permutation WCSP model from an existing permutation WCSP using *generalized model induction*. We then uncover why naively combining mutually redundant permutation WCSPs by posting channeling constraints as hard constraints and relying on the standard *node consistency* (NC*) and *arc consistency* (AC*) algorithms would miss pruning opportunities, which are available even in a single model. Based on these observations, we suggest two approaches to handle the combined WCSP models. In our first approach, we propose m-NC^*_c and m-AC^*_c and their associated algorithms for effectively enforcing node and arc consistencies in a combined model with m sub-models. The two notions are strictly stronger than NC* and AC* respectively. While the first approach specifically refines NC* and AC* so as to apply to combined models, in our second approach, we propose a parameterized local consistency LB(m,Φ). The consistency can be instantiated with *any* local consistency Φ for single models and applied to a combined model with m sub-models. We also provide a simple algorithm to enforce LB(m,Φ). With the two suggested approaches, we demonstrate their applicabilities on several permutation problems in the experiments. Prototype implementations of our proposed algorithms confirm that applying 2-NC^*_c, 2-AC^*_c, and LB(2,Φ) on combined models allow far more constraint propagation than applying the state-of-the-art AC*, FDAC*, and EDAC* algorithms on single models of hard benchmark problems.

1 Introduction

Many decision problems can be modeled as *constraint satisfaction problems* (CSPs) [35]. Some examples are map coloring, job-shop scheduling [17, 16], resource allocation [41, 9], and machine vision [50]. However, the classical framework of CSPs is not designed to model optimization problems and over-constrained problems, where there are costs, preferences, or no solutions. Thus, soft constraint frameworks are proposed to model these problems by allowing a degree of constraint violation. *Weighted CSPs* (WCSPs) [43, 26, 27, 28, 11] form a common soft constraint framework which extends the CSP framework by associating costs to variable assignments [44]. We can then give preferences to variable assignments by specifying their costs.

Given a problem, there are usually more than one way to model the problem as a CSP. We can connect two different CSP models of a problem using *channeling constraints*, which are constraints relating the variable assignments of the two constituent models. This modeling technique is known as *redundant modeling* [5]. Redundant modeling has been shown effective in increasing constraint propagation and hence reducing search efforts for solving CSPs. While the technique has been applied successfully to classical CSPs, in this paper we investigate, for the first time, how to benefit the same for *weighted CSPs*.

In order to apply the redundant modeling technique, we first need to obtain two mutually redundant models of the same problem. With classical CSPs, this can be done relatively easily by human modelers: we simply need to ensure that each problem requirement is represented in both models. In the WCSP framework, however, each problem solution is associated with a cost. Therefore, obtaining mutually redundant models is more difficult in general, since besides the problem requirements, we also need to ensure the same cost distribution on the solutions of the two models. We resolve this problem by generalizing *model induction* [29, 30], a method which can automatically generate a redundant CSP from a given one, so that a redundant WCSP can also be obtained. Since the original model induction is applicable to only permutation CSPs in practice, we focus our work on generating and combining *permutation WCSPs* as well.

Two mutually redundant models can be combined using channeling constraints. For classical CSPs, we can rely on the standard propagation algorithms of the channeling constraints to transmit pruning information between sub-models to achieve stronger propagation. We can also do the same to combine WCSPs by posting the channeling constraints as hard constraints. However, we discover that applying the standard *star node consistency* (NC*) and *star arc consistency* (AC*) algorithms [26] to a combined model does not increase constraint propagation. In fact, some prunings that are available when propagating a single model alone would even be missed in a combined model, resulting in worse performance. Based on these observations, we suggest *two* approaches to handle the combined WCSP models.

In our first approach, we *generalize* the notions of node and arc consistencies and *propose* m-NC$^*_c$ and m-AC$^*_c$ for a combined model with m sub-models. The m-NC$^*_c$ (*resp.* m-AC$^*_c$) notion is strictly stronger than NC* (*resp.* AC*), and degenerates to NC* (*resp.* AC*) when $m = 1$. This approach is limited to NC* and AC*, which is not applicable if one wants to use other local consistencies to solve a problem. Thus, we have our second approach, which does not require any modification of a particular local consistency. In our second approach, we proposed a *parameterized* local consistency LB(m,Φ) and its associated enforcement algorithm. The advantages of this approach is three-fold. First, the LB(m,Φ) consistency can be instantiated with *any* local consistency Φ for single models and can be applied to a combined model with m sub-models. Second, the local consistency Φ used for instantiation needs not be refined. Third, enforcing LB(m,Φ) on a combined WCSP model $\mathcal{P}$ achieves *stronger* constraint propagation than enforcing Φ on any individual sub-model of $\mathcal{P}$ alone. Both approaches can be applied to combined models to restore the missed prunings and even discover new possible prunings to enhance constraint propagation. We test our implementations using both classical and soft benchmark problems. The soft problems are obtained from the classical ones using two common problem softening schemes. Experimental results confirm that the proposed algorithms are particularly useful to solve hard problem instances, and significantly reduce the search space and runtime over enforcing the state-of-the-art soft local consistencies AC* [26], FDAC* [27], and EDAC* [11] on single models. Although we show that m-AC$^*_\text{c}$ is strictly stronger than LB(m,Φ) and achieves more prunings in problem solving, LB(m,Φ) is competitive with m-AC$^*_\text{c}$ due to its flexibility of choosing any local consistency and less overhead.

This paper, a revised and extended version of the work by Law, Lee, and Woo [31, 32], gives a comprehensive report on using *redundant modeling in permutation WCSPs*. The paper is organized as follows. Section 2 provides the background to the paper. We formally define the concepts of WCSPs and present how common consistency techniques can be incorporated into branch and bound search to improve solving efficiency. Section 3 introduces permutation WCSP and describes how we can generate a redundant permutation WCSP model based on the idea of model induction [29, 30]. In Section 4, we first show how two redundant models can be combined using channeling constraints. We then explain why applying NC* and AC* to a combined model can result in weaker propagation than that to a single model, and propose m-NC$^*_\text{c}$ and m-AC$^*_\text{c}$, which adapt the existing NC* and AC* algorithm, to resolve the problem (first approach). Algorithms for enforcing m-NC$^*_\text{c}$ and m-AC$^*_\text{c}$ are also presented. Section 5 describes another method to combine two redundant models and gives the notion of a parameterized local consistency LB(m,Φ) (second approach). A simple algorithm to enforce LB(m,Φ) is also provided. Theoretical comparisons between the consistencies used in the first and the second approaches, and the semantics between the combined models and their sub-models are discussed. Section 6 presents our experimental results on both classical and soft benchmark problems to evaluate the performance of 2-AC$^*_\text{c}$, LB(2,AC*), LB(2,FDAC*), and LB(2,EDAC*) on combined models against various consistencies on single models. Section 7 gives an overview of the related work on soft constraints, local consistencies in WCSP, and redundant modeling. Section 8 concludes this paper by summarizing our contributions and giving possible directions for future research.

2 Background

In this section, we provide the definitions of some terminologies relating to WCSPs and briefly describe redundant modeling.

2.1 Weighted Constraint Satisfaction Problems

Weighted CSPs (WCSPs) [43, 26, 27, 28, 11] associate *costs* to tuples [44]. The costs are specified by a *valuation structure* $S(k)$. A *valuation structure* $S(k)$ is a triple $([0,\ldots,k],\oplus,\geq)$ where $k \in \{1,\ldots,\infty\}$ is either a strictly positive natural number or infinity, $\oplus$ is defined as $a \oplus b = \min\{k, a+b\}$, and $\geq$ is the standard order among natural numbers. The minimum and maximum costs are denoted by the *bottom* $\bot = 0$ and *top* $\top = k$ respectively.

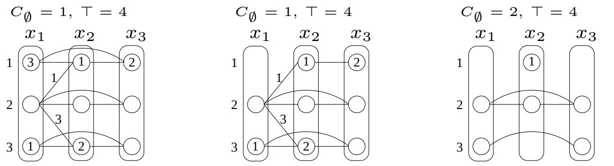
A binary WCSP is a quadruplet $\mathcal{P} = (k, \mathcal{X}, \mathcal{D}, \mathcal{C})$ with the valuation structure $S(k)$. $\mathcal{X} = \{x_1,\ldots,x_n\}$ is a finite set of *variables* and $\mathcal{D} = \{D_{x_1},\ldots,D_{x_n}\}$ is a set of finite *domains* for each $x_i \in \mathcal{X}$. Following Law and Lee [30], we define a pair $(\mathcal{X}, \mathcal{D})$ as a *viewpoint* of the WCSP. An *assignment* $x_i \mapsto a$ in $\mathcal{P}$ is a mapping from variable x_i to value $a \in D_{x_i}$. A *tuple* is a set of assignments in $\mathcal{P}$. It is *complete* if it contains assignments of all variables in $\mathcal{P}$. $\mathcal{C}$ is a set of unary and binary constraints and a zero-arity constraint. A *unary constraint* involving variable x_i in $\mathcal{P}$ is a cost function $C_{x_i} : D_{x_i} \to \{0,\ldots,k\}$ which assigns costs to assignments of x_i. A *binary constraint* involving variables x_i and x_j in $\mathcal{P}$ is a cost function $C_{x_i,x_j} : D_{x_i} \times D_{x_j} \to \{0,\ldots,k\}$ which assigns costs to tuples of x_i and x_j. Following Freduder and Wallace [18], we also assume a *zero-arity* constraint $C_\emptyset$ [26, 27, 28] in $\mathcal{P}$ which is a constant and is set to $\bot$. As we shall see, $C_\emptyset$ can be increased during WCSP solving to denote the global lower bound of costs in $\mathcal{P}$. We also assume without loss of generality that there is a binary constraint for each variable pair and a unary constraint for each variable.

We denote the set of variables in a tuple θ as $var(\theta)$. The *projection* of a tuple θ over a variable subset $U \subseteq var(\theta)$ is defined as $\theta_{\downarrow U} = \{(x_i \mapsto a) \in \theta \mid x_i \in U\}$, which is a subset of assignments of θ whose variables are in U. The cost $\mathcal{V}(\theta)$ of a tuple θ in $\mathcal{P}$ is the sum of all applicable costs, i.e.,

$$\mathcal{V}(\theta) = C_\emptyset \oplus \sum_{x_i \in var(\theta)} C_{x_i}(\theta \downarrow_{\{x_i\}}) \oplus \sum_{x_i,x_j \in var(\theta)} C_{x_i,x_j}(\theta \downarrow_{\{x_i,x_j\}}).$$

A tuple θ is *consistent* if $\mathcal{V}(\theta) < \top$. A *solution* of $\mathcal{P}$ is a consistent complete tuple. Solving a WCSP $\mathcal{P}$ is to find an *optimal solution* of $\mathcal{P}$, which is a solution θ of $\mathcal{P}$ with minimum $\mathcal{V}(\theta)$. WCSP solving is NP-hard. Two WCSPs are *equivalent* if they have the same variables and for every complete tuple θ, $\mathcal{V}(\theta)$ is the same for both WCSPs.

Example 2.1 *Figure 1(a) shows a WCSP with variables* $\{x_1, x_2, x_3\}$ *and domains* $\{1, 2, 3\}$. *We depict the unary costs as labeled nodes and binary costs as*



(a) $\mathcal{P}$ (b) $\mathcal{P}$ after NC* (c) $\mathcal{P}$ after AC*

Figure 1: Three equivalent WCSPs

labeled edges connecting two assignments. Unlabeled edges have $\top$ *cost;* $\bot$ *costs are not shown for clarity. The WCSP has a minimum cost* 3 *with the solution* $\{x_1 \mapsto 2, x_2 \mapsto 1, x_3 \mapsto 3\}$ *(*$C_\emptyset \oplus C_{x_1}(2) \oplus C_{x_2}(1) \oplus C_{x_3}(3) \oplus C_{x_1,x_2}(2,1) \oplus C_{x_1,x_3}(2,3) \oplus C_{x_2,x_3}(1,3) = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 = 3$*).* ■

2.2 Local Consistencies

WCSPs can be solved by backtracking *branch and bound* (B&B) search [25] that maintains some form of local consistency at each search node. A *local consistency* is simply some properties of a WCSP. *Enforcing* a local consistency Φ on a WCSP $\mathcal{P}$ means transforming $\mathcal{P}$ to an equivalent WCSP $\mathcal{P}'$ that satisfies the properties specified by Φ. $\mathcal{P}'$ is usually simplified in the sense that either some domain values of the variables are pruned or the lower bound $C_\emptyset$ is increased. In the following, we give the definitions of two common local consistencies NC* [26, 28] and AC* [43, 26, 28] for WCSPs. They degenerate to their standard counterparts in CSPs.

Definition 2.1 *Let* $\mathcal{P} = (k, \mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X})$ *be a binary WCSP.*

Node consistency *An assignment* $x \mapsto a$ *in* $\mathcal{P}$ *is* star node consistent *(NC*) if* $C_\emptyset \oplus C_x(a) < \top$*. A variable* x *in* $\mathcal{P}$ *is NC* if (1) all assignments of* x *are NC* and (2) there exists an assignment* $x \mapsto a$ *of* x *such that* $C_x(a) = \bot$*. Value* a *is a* support *for* x*.* $\mathcal{P}$ *is NC* if every variable in* $\mathcal{P}$ *is NC*.*

Arc consistency *An assignment* $x_i \mapsto a$ *in* $\mathcal{P}$ *is* arc consistent *(AC) with respect to a constraint* C_{x_i,x_j} *if there exists an assignment* $x_j \mapsto b$ *of* x_j *such that* $C_{x_i,x_j}(a,b) = \bot$*. Value* b *is a* support *for* $x_i \mapsto a$*. A variable* x_i *in* $\mathcal{P}$ *is AC if all assignments of* x_i *are AC with respect to all binary constraints involving* x_i*.* $\mathcal{P}$ *is* star arc consistent *(AC*) if every variable in* $\mathcal{P}$ *is NC* and AC.*

NC* can be enforced by *projections* of unary constraints over $C_\emptyset$ and pruning node inconsistent values [43, 26, 28]. Let $0 \le b \le a \le k$ be two costs. *Subtraction* of b from a is defined as:

$$a \ominus b = \begin{cases} a - b & \text{if } a \neq k \\ k & \text{otherwise.} \end{cases}$$

Consider a variable x_i in a WCSP $\mathcal{P}$, let $\alpha = \min_{a \in D_{x_i}} \{C_{x_i}(a)\}$ be the minimum cost incurred by C_{x_i}. Freuder and Wallace [18] suggested that α is a necessary cost for any complete tuple of $\mathcal{P}$. Using this idea, (unary) *projection* of C_{x_i} over $C_\emptyset$ [26, 28] is defined as a flow of α cost units such that (1) $C_\emptyset := C_\emptyset \oplus \alpha$, and (2) for each $a \in D_{x_i}$, $C_{x_i}(a) := C_{x_i}(a) \ominus \alpha$. The updated $C_\emptyset$ becomes the new global lower bound of the cost of the problem $\mathcal{P}$. After forcing supports for all variables, all assignments $x_i \mapsto a$ with $C_\emptyset \oplus C_{x_i}(a) = \top$ can be removed. NC* can be enforced in $O(nd)$ time, where n is the number of variables and d is the maximum domain size, using the NC* algorithm [26, 28].

Similarly, AC* can be enforced by *projections* of binary constraints over unary constraints, and then relying on NC* to further move the costs to $C_\emptyset$ or prune any values [43, 26, 28]. Given variables x_i and x_j, for each $a \in D_{x_i}$, let $\alpha_a = \min_{b \in D_{x_j}} \{C_{x_i,x_j}(a, b)\}$ be the minimum cost of $x_i \mapsto a$ incurred by the binary constraint C_{x_i,x_j}. (Binary) *projection* of C_{x_i,x_j} over C_{x_i} [43, 26, 28] is defined such that (1) for each $a \in D_{x_i}$, $C_{x_i}(a) := C_{x_i}(a) \oplus \alpha_a$, and (2) for each $a \in D_{x_i}$ and $b \in D_{x_j}$, $C_{x_i,x_j}(a, b) := C_{x_i,x_j}(a, b) \ominus \alpha_a$. It thus transforms a WCSP $\mathcal{P}$ into an equivalent WCSP $\mathcal{P}'$ [43]. AC* can be enforced in $O(n^2d^3)$ time using the AC* algorithm [26, 28], which is basically repeated applications of unary and binary projections until the WCSP becomes quiescent.

Example 2.2 *The WCSP in Figure 1(a) is not NC* since $C_\emptyset \oplus C_{x_1}(1) = 1 \oplus 3 = 4 = \top$. Removing value 1 from D_{x_1} makes it NC* and equivalent to the one in Figure 1(a). However, it is still not AC*. Consider the assignment $x_1 \mapsto 2$ and the constraint C_{x_1,x_2} in Figure 1(b), all the costs $C_{x_1,x_2}(2, 1)$, $C_{x_1,x_2}(2, 2)$, and $C_{x_1,x_2}(2, 3)$ are non-$\perp$. We can subtract 1 from each of these costs and add 1 to $C_{x_1}(2)$ to force a support for $x_1 \mapsto 2$. The WCSP in Figure 1(c) is AC* and equivalent to the one in Figure 1(b).* ■

In backtracking B&B search, whenever $C_\emptyset$ is increased to $\top$, we cannot continue to extend a tuple to obtain a solution, and hence a backtrack (or sometimes called a fail) is triggered. Also, whenever a solution θ is found, the $\top$ value will be set to $\mathcal{V}(\theta)$ to continue the search, ensuring that the next solution found must have a better cost than θ. At the end of the search, the last found solution is optimal.

3 Generating Redundant Permutation WCSPs

Deriving multiple classical CSP models for the same problem is common, although not trivial. Law and Lee [29, 30] proposed *model induction* which automatically generates a redundant CSP from a given one. However, it is even more difficult to obtain an alternative model in WCSP since each problem solution is associated with a cost and we have to ensure the same cost distribution on the solutions of the redundant WCSP models. Two WCSPs $\mathcal{P}_1$ and $\mathcal{P}_2$ are *mutually redundant* if (1) there is a bijection g between the two sets of all solutions of $\mathcal{P}_1$ and $\mathcal{P}_2$, and (2) for every solution θ of $\mathcal{P}_1$, the associated costs of solution θ of $\mathcal{P}_1$ and solution $g(\theta)$ of $\mathcal{P}_2$ are the same, i.e., $\mathcal{V}(\theta) = \mathcal{V}(g(\theta))$. Though the operation is a general one, we propose a slight generalization of model induction that generates mutually redundant *permutation* WCSPs from a given one. We focus on permutation WCSPs because their redundant models can be naturally and easily obtained.

3.1 Permutation WCSPs

A *permutation WCSP* is a WCSP $\mathcal{P} = (k, \mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X})$ in which each variable takes a unique value and $|\mathcal{X}|$ is the same as the size of the variable domains. Any solution of $\mathcal{P}$ assigns a permutation of the domain values to the variables. In addition, the all-different constraints in the permutation WCSP must be hard constraints which means that tuples having assignments of the same value to different variables have a $\top$ cost, i.e., $C_{x_i,x_j}(a, a) = \top$ for all variables x_i, x_j and $a \in D_{x_i} = D_{x_j}$. For other constraints, they can have a cost between $\perp$ and $\top$. Given a permutation WCSP $\mathcal{P} = (k, \mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X})$ with $\mathcal{X} = \{x_1, \ldots, x_n\}$, we can always interchange the roles of its variables and values to give a *dual* permutation WCSP $\mathcal{P}' = (k, \mathcal{Y}, \mathcal{D}_\mathcal{Y}, \mathcal{C}_\mathcal{Y})$ with $\mathcal{Y} = \{y_1, \ldots, y_n\}$. The variables in $\mathcal{X}$ and $\mathcal{Y}$ can be related using the channeling constraints $x_i \mapsto j \Leftrightarrow y_j \mapsto i$ for $1 \leq i, j \leq n$.

Example 3.1 *Recall the WCSP in Figure 9(a). It is a permutation WCSP which has three variables and each variable has domain values* $\{1, 2, 3\}$*. The all-different constraints* $x_i \neq x_j$*, where* $1 \leq i < j \leq 3$*, have cost functions* $C_{x_i,x_j}(a, a) = \top$*, where* $a \in \{1, 2, 3\}$*. There are other constraints in which the binary costs are between* $\perp$ *and* $\top$*.* ■

3.2 Generalized Model Induction for Permutation WCSPs

Given a classical CSP, we can always model it as a WCSP in which the compatible assignments are of cost $\perp$ and the incompatible ones are of cost $\top$. However, constraints in WCSPs are soft: they can have costs between $\perp$ and $\top$. For two mutually redundant WCSP models, there must exist a bijective mapping between the two sets of all solutions. Besides, we have to ensure the same cost between every pair of equivalent solutions in the two WCSP models.

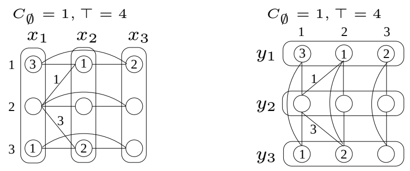
Based on these two requirements, we proposed *generalized model induction* that generates mutually redundant *permutation WCSPs* from a given one, using another viewpoint and channeling constraints. In a permutation WCSP, the variables in a solution must take all-different values. Given a WCSP $\mathcal{P} = (k, \mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X})$, a *channel function* maps assignments in $\mathcal{P}$ to those in another set of variables. If $\mathcal{P}$ is a permutation WCSP, without loss of generality, we always have the bijective channel function $f(x_i \mapsto j) = y_j \mapsto i$. The constraints $\mathcal{C}_\mathcal{Y}$ in the *induced model* $\mathcal{P}' = (k, \mathcal{Y}, \mathcal{D}_\mathcal{Y}, \mathcal{C}_\mathcal{Y})$ are defined such that

$$
\begin{array}{rcl}
C_{y_a}(i) & = & C_{x_i}(a) \qquad\qquad\qquad \text{for } 1 \leq a, i \leq n \\
C_{y_a,y_b}(i, j) & = & \left\{ \begin{array}{cl} C_{x_i,x_j}(a, b) & \text{if } i \neq j \text{, for } 1 \leq a, b, i, j \leq n \\ \top & \text{if } i = j \text{, for } 1 \leq a, b, i, j \leq n \end{array} \right. .
\end{array}
$$

Note that the induced model $\mathcal{P}'$ must be a permutation WCSP, since $C_{y_a,y_b}(i,i) = \top$ for all $1 \le a, b, i \le n$.

Theorem 3.1 *Given a permutation WCSP $\mathcal{P} = (k, \mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X})$ and another viewpoint $(\mathcal{Y}, \mathcal{D}_\mathcal{Y})$ such that $(\mathcal{X}, \mathcal{D}_\mathcal{X})$ and $(\mathcal{Y}, \mathcal{D}_\mathcal{Y})$ can be connected by the channeling constraints $x_i = j \Leftrightarrow y_j = i$ for all $1 \le i, j, \le n$, where $n = |\mathcal{X}| = |\mathcal{Y}|$. $\mathcal{P}$ and its induced model $\mathcal{P}'$ are mutually redundant WCSP models.*

Proof 3.1 *In generalized model induction, an assignment $x_i \mapsto a$ in $\mathcal{P}$ can always be mapped to an assignment $y_a \mapsto i$ in $\mathcal{P}'$ through the bijective function $f(x_i \mapsto a) = y_a \mapsto i$. Thus, a solution of $\mathcal{P}$, which is a complete tuple θ_1, can also be mapped correspondingly to a solution θ_2 in $\mathcal{P}'$ and there is a bijective mapping between the two sets of solutions of $\mathcal{P}$ and $\mathcal{P}'$. Given two equivalent solutions $\theta = \{x_{v_1} \mapsto 1, \ldots, x_{v_n} \mapsto n\}$ in $\mathcal{P}$ and $\theta' = \{y_1 \mapsto v_1, \ldots, y_n \mapsto v_n\}$ in $\mathcal{P}'$ through the bijection, according to generalized model induction, each assignment $(x_{v_i} \mapsto i) \in \theta$ has the same unary cost as the corresponding assignment $(y_i \mapsto v_i) \in \theta'$ (i.e., $C_{x_{v_i}}(i) = C_{y_i}(v_i)$). Similarly, every pair of assignments $\{x_{v_i} \mapsto i, x_{v_j} \mapsto j\}$ in $\mathcal{P}$ has the same binary cost as the corresponding pair of assignments $\{y_i \mapsto v_i, y_j \mapsto v_j\}$ in θ (i.e., $C_{x_{v_i},x_{v_j}}(i,j) = C_{y_i,y_j}(v_i,v_j)$). The associated cost of the solution θ in $\mathcal{P}$ equals that of θ' in $\mathcal{P}'$, i.e., $\mathcal{V}(\theta) = \mathcal{V}(\theta')$. Therefore, we satisfy the two conditions of mutual redundancy, and $\mathcal{P}$ and $\mathcal{P}'$ are mutually redundant WCSP models.* $\square$



(a) $\mathcal{P}_1$ (b) $\mathcal{P}_2$

Figure 2: An example of model induction of a permutation WCSP

Example 3.2 *Consider the WCSP $\mathcal{P}_1 = (4, \mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X})$ in Figure 2(a). In $\mathcal{P}_1$, the valuation structure is $S(4)$ and we have variables $\mathcal{X} = \{x_1, x_2, x_3\}$. Each variable has domain values $D_{x_i} = \{1, 2, 3\}$. In generalized model induction, the role of variables and values are interchanged to generate an induced model as shown in Figure 2(b).*

In the induced model, the valuation structure remains $S(4)$, but we now have variables $\{y_1, y_2, y_3\}$, each of which has the same set of domain values $D_{y_i} = \{1, 2, 3\}$. Both $\mathcal{P}_1$ and $\mathcal{P}_2$ have the same values for $C_\emptyset$ and $\top$. The unary and binary costs are also transformed from $\mathcal{P}_1$ to $\mathcal{P}_2$ correspondingly. For example, the unary cost $C_{x_2}(1) = 1$ in $\mathcal{P}_1$ becomes a unary cost $C_{y_1}(2) = 1$ in $\mathcal{P}_2$. For the binary cost $C_{x_1,x_2}(2,3) = 3$ in $\mathcal{P}_1$, we transform it into the binary cost $C_{y_2,y_3}(1,2) = 3$ in $\mathcal{P}_2$. Besides, the all-different constraints, such as $C_{y_1,y_2}(1,1) = \top$, are added to $\mathcal{P}_2$. ■

4 Refining Local Consistency

In this section, we introduce our first approach to combine and solve two mutually redundant permutation WCSP models. We give detailed explanations to show the problems encountered when we simply apply the original notions of NC* and AC* on a combined model. Based on the investigation, we refine the definitions of node and arc consistencies to give m-NC$^*_{\text{c}}$ and m-AC$^*_{\text{c}}$, which are applicable to a combined model with m sub-models. Their respective enforcing algorithms are also given.

4.1 Naive Approach

Given two mutually redundant (classical) CSPs $\mathcal{P}_1 = (\mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X})$ and $\mathcal{P}_2 = (\mathcal{Y}, \mathcal{D}_\mathcal{Y}, \mathcal{C}_\mathcal{Y})$, a combined model is a CSP $\mathcal{P}^c = (\mathcal{X} \cup \mathcal{Y}, \mathcal{D}_\mathcal{X} \cup \mathcal{D}_\mathcal{Y}, \mathcal{C}_\mathcal{X} \cup \mathcal{C}_\mathcal{Y} \cup \mathcal{C}^c)$, where $\mathcal{C}^c$ is the set of channeling constraints connecting $\mathcal{P}_1$ and $\mathcal{P}_2$, the *sub-models* of $\mathcal{P}^c$. *Channeling constraints* define the relationship between the variable assignments in the two sub-models. They are typically of the form $x_i \mapsto a \Leftrightarrow y_j \mapsto b$ where $x_i \mapsto a$ and $y_j \mapsto b$ are assignments of $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively. The channeling constraints $x_i \mapsto j \Leftrightarrow y_j \mapsto i$ connecting a permutation CSP and its dual model are of this form.

We can construct a combined model $\mathcal{P}^c$ for two mutually redundant WCSPs $\mathcal{P}_1 = (k_1, \mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X})$ and $\mathcal{P}_2 = (k_2, \mathcal{Y}, \mathcal{D}_\mathcal{Y}, \mathcal{C}_\mathcal{Y})$ similarly. $\mathcal{P}^c$ has the valuation structure $S(k_1 + k_2)$. The optimum of $\mathcal{P}^c$ is twice the optimum of the original models $\mathcal{P}_1$ and $\mathcal{P}_2$ (i.e., $\mathcal{V}(\theta) = 2 \cdot \mathcal{V}(\theta_1) = 2 \cdot \mathcal{V}(\theta_2)$). The variables, domains, and constraints are formed like in the classical case. The channeling constraints are included in $\mathcal{P}^c$ as hard constraints, i.e., all costs are either $\perp$ or $\top$. For permutation WCSPs, every channeling constraint $x_i \mapsto j \Leftrightarrow y_j \mapsto i$ has the cost function

$$C_{x_i, y_j}(a, b) = \begin{cases} \perp & \text{if } a = j \Leftrightarrow b = i \\ \top & \text{otherwise.} \end{cases}$$

Example 4.1 *Recall the permutation WCSP model $\mathcal{P}_1$ and its induced WCSP model $\mathcal{P}_2$ in Figure 2. We use those two models to illustrate how they are combined to form a combined model $\mathcal{P}^c$.*

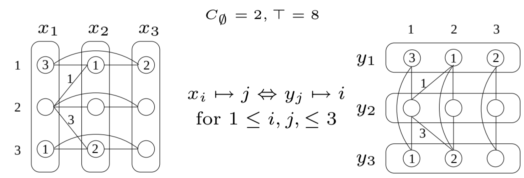


Figure 3: An example of a combined permutation WCSP model

In the combined model $\mathcal{P}^c$, the valuation structure is $S(4+4) = S(8)$. The variables and domains are $\mathcal{X} \cup \mathcal{Y}$ and $\mathcal{D}_\mathcal{X} \cup \mathcal{D}_\mathcal{Y}$ respectively. The set of constraints in $\mathcal{P}^c$ is a combination of the constraints in $\mathcal{P}_1$ and $\mathcal{P}_2$, and the channeling constraints $\mathcal{C}^c = \{x_i \mapsto j \Leftrightarrow y_j \mapsto i \,|\, 1 \leq i, j \leq 3\}$. The costs of the *channeling constraints are either* $\perp$ *or* $\top$*. For example, the channeling constraint* $x_2 \mapsto 1 \Leftrightarrow y_1 \mapsto 2$ *has the cost function:*

$$\begin{array}{lll} C_{x_2,y_1}(1,1) = \top, & C_{x_2,y_1}(1,2) = \perp, & C_{x_2,y_1}(1,3) = \top, \\ C_{x_2,y_1}(2,1) = \perp, & C_{x_2,y_1}(2,2) = \top, & C_{x_2,y_1}(2,3) = \perp, \\ C_{x_2,y_1}(3,1) = \perp, & C_{x_2,y_1}(3,2) = \top, & C_{x_2,y_1}(3,3) = \perp. \end{array}$$

Thus, the combined WCSP model is a quadruplet $\mathcal{P}^c = (8, \mathcal{X} \cup \mathcal{Y}, \mathcal{D}_\mathcal{X} \cup \mathcal{D}_\mathcal{Y}, \mathcal{C}_\mathcal{X} \cup \mathcal{C}_\mathcal{Y} \cup \mathcal{C}^c)$*. Figure 3 shows the combined model* $\mathcal{P}^c$ *for* $\mathcal{P}_1$ *and* $\mathcal{P}_2$*. For clarity, the edges of the channeling constraints are not shown in the figure. The set of assignments* $\{x_1 \mapsto 2, x_2 \mapsto 1, x_3 \mapsto 3, y_1 \mapsto 2, y_2 \mapsto 1, y_3 \mapsto 3\}$ *is one of the solutions in* $\mathcal{P}^c$*. Its cost doubles the costs of the corresponding assignments in* $\mathcal{P}_1$ *and* $\mathcal{P}_2$*.* ■

Table 1: Preliminary experimental results on solving soft Langford's problem

	AC* on $\mathcal{P}$		AC* on $\mathcal{P}^c$	
(m,n)	fail	time	fail	time
$(3,11)$	77507	43.8	68342	163.16
$(3,12)$	275643	178.25	172520	546.96

We perform some preliminary experiments in the WCSP solver ToolBar [4] using Langford's problem (prob024 in CSPLib [19]) to evaluate the performance of the single and combined models. The Langford's problem, denoted as (m, n)-Langford's problem, is to find an $m \times n$ digit sequence consisting of digits 1 to n, each occurring m times, such that any two consecutive occurrences of digit i's are separated by i other digits. For example, a solution of the $(2, 3)$ instance is 312132. The single model $\mathcal{P}$ is based on a permutation CSP model suggested by Hnich, Smith, and Walsh [22]. This problem is over-constrained for many instances. For example, only $(3, 9)$ and $(3, 10)$ among the $(3, n)$ instances for $3 \le n \le 16$ have solutions. Therefore, we soften the model so that tuples of the all-different constraints remains hard, and other constraints in the model can now have random non-$\top$ costs. With the aid of generalized model induction, we obtain an induced model from an original one. The combined model $\mathcal{P}^c$ contains the single model and its induced model as sub-models. We randomly generate 10 models for each of the $(3, 11)$ and $(3, 12)$ instances and obtain the average results of the number of fails (i.e., the number of backtracks occurred in solving a model) and CPU time in seconds in Table 1 for finding the first optimal solution.

From the result, we find that enforcing AC* on a combined model does achieve fewer number of fails than enforcing AC* on a single model. However, execution of the former takes much longer time than that of the latter. This is mainly due to three reasons. First, there are more variables and constraints in a combined model than in a single model. It takes longer time to propagate the constraints at each node in a search tree. Second, there is a large number of channeling constraints connecting the two mutually redundant models. For example, in (m, n)-Langford's problem, we need m^2n^2 channeling constraints to combine two mutually redundant models. For classical CSPs, efficient global constraints exist for propagating the channeling constraints, but there are no such counterparts for WCSPs. Third, by analyzing the propagation behavior in the combined models, we find that despite achieving fewer number of fails overall, enforcing AC* on a combined model can miss pruning opportunities which are available even in a single model. We shall discuss in details why this can happen, and as a remedy to the second and third drawbacks, we propose m-NC$^*_\mathrm{c}$ and m-AC$^*_\mathrm{c}$ and their associated algorithms for effectively improving propagation in a combined model with m sub-models. We also reveal that the propagation of pruning information among sub-models can be done by enforcing m-NC$^*_\mathrm{c}$. This means that redundant modeling can be done without the channeling constraints.

4.2 Node Consistency Revisited

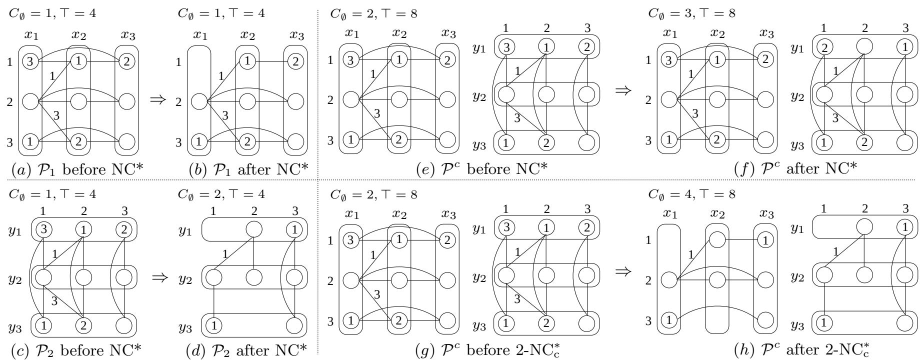


Figure 4: Enforcing node consistencies on $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}^c$

We first investigate the problems encountered when enforcing NC* in the combined models. Figures 4(a) and 4(c) show two mutually redundant WCSPs $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, as given in Figures 2(a) and 2(b). Figure 4(e) gives the combined model $\mathcal{P}^c$, which is the same as the one shown in Figure 3.

Consider enforcing NC* on $\mathcal{P}_1$ and $\mathcal{P}_2$ individually. The assignment $x_1 \mapsto 1$ in $\mathcal{P}_1$ is not NC*, since $C_\emptyset \oplus C_{x_1}(1) = 1 \oplus 3 = \top$. Value 1 can hence be removed from D_{x_1}, as shown in Figure 4(b). Similarly, value 1 can be also removed from D_{y_1}. Furthermore, neither of the remaining domain values $\{2, 3\}$ of y_1 has a $\perp$ unary cost. Therefore, C_{y_1} is projected over $C_\emptyset$ such that $C_{y_1}(2) = \perp$, $C_{y_1}(3) = 1$, and $C_\emptyset = 2$. After increasing $C_\emptyset$, the assignment $y_2 \mapsto 2$ in $\mathcal{P}_2$ is no longer NC*, since $C_\emptyset \oplus C_{y_2}(2) = 2 \oplus 2 = \top$. Value 2 is thus removed from D_{y_2}, resulting an equivalent WCSP in Figure 4(d) which is NC*.

Now consider enforcing NC* on the combined model $\mathcal{P}^c$ of $\mathcal{P}_1$ and $\mathcal{P}_2$. $\mathcal{P}^c$ has a valuation structure $S(4 + 4) = S(8)$. In $\mathcal{P}^c$, however, $x_1 \mapsto 1$ and $y_1 \mapsto 1$ are still NC*, since $C_\emptyset \oplus C_{x_1}(1) = C_\emptyset \oplus C_{y_1}(1) = 5 < \top = 8$. Therefore, no values can be removed from the variable domains. Enforcing NC* on $\mathcal{P}^c$ can only project C_{y_1} over $C_\emptyset$ such that $C_{y_1}(1) = 2$, $C_{y_1}(2) = \bot$, $C_{y_1}(3) = 1$, and $C_\emptyset = 3$, as shown in Figure 4(f). This example shows the undesirable behavior that enforcing NC* on a combined model can miss pruning opportunities that are available even in single models, resulting in weaker constraint propagation than on its sub-models individually.

4.2.1 Refining the Node Consistency Definition

From the previous example, we observe that given any solution θ of the combined model $\mathcal{P}^c$, if an assignment $x_i \mapsto j$ in sub-model $\mathcal{P}_1$ is in θ, then according to the channeling constraints $x_i \mapsto j \Leftrightarrow y_j \mapsto i$, the corresponding assignment $y_j \mapsto i$ in sub-model $\mathcal{P}_2$ must be also in θ, and vice versa. Therefore, we can check the consistencies of $x_i \mapsto j$ and $y_j \mapsto i$ *simultaneously*. If the global lower bound $C_\emptyset$ plus the sum of the unary costs $C_{x_i}(j)$ and $C_{y_j}(i)$ equals $\top$, then both $x_i \mapsto j$ and $y_j \mapsto i$ cannot be in any solution of $\mathcal{P}^c$ and can be pruned. For example, consider assignments $x_1 \mapsto 1$ and $y_1 \mapsto 1$ in Figure 4(f). Since $C_\emptyset \oplus C_{x_1}(1) \oplus C_{y_1}(1) = 3 \oplus 3 \oplus 2 = 8 = \top$, both $x_1 \mapsto 1$ and $y_1 \mapsto 1$ should be pruned, thus restoring the available prunings in the single models.

Furthermore, consider variable y_1 in $\mathcal{P}_2$, a complete tuple of $\mathcal{P}_2$ must contain exactly one assignment of y_1. The set of assignments $\{y_1 \mapsto 1, y_1 \mapsto 2, y_1 \mapsto 3\}$ in $\mathcal{P}_2$ correspond to $\theta = \{x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 1\}$ in $\mathcal{P}_1$. Therefore, a solution of $\mathcal{P}^c$ must contain exactly one assignment among θ. In Figure 4(f), since the minimum cost among $C_{x_1}(1)$, $C_{x_2}(1)$, and $C_{x_3}(1)$ is $1 > \bot$, we can use such information to tighten the global lower bound of $\mathcal{P}^c$ in addition to the projection of C_{y_1} over $C_\emptyset$.

By capturing the ideas described in the previous two paragraphs, we propose a new notion of node consistency m-NC$^*_\mathrm{c}$ for combined WCSP models with m sub-models. Note that m-NC$^*_\mathrm{c}$ is a general notion; it is not restricted to permutation WCSPs only. In the following, we assume that $\mathcal{P}_s = (k_s, \mathcal{X}_s, \mathcal{D}_s, \mathcal{C}_s)$ for $1 \le s \le m$ are m mutually redundant WCSPs, where $\mathcal{X}_s = \{x_{s,i} \,|\, 1 \le i \le n_s\}$ and $\mathcal{D}_s = \{D_{x_{s,i}} \,|\, 1 \le i \le n_s\}$ ($n_s = |\mathcal{X}_s|$). $\mathcal{C}_{s,t}$ is the set of channeling constraints connecting $\mathcal{P}_s$ and $\mathcal{P}_t$, and $\mathcal{C}^c = \bigcup_{1 \le s < t \le m} C_{s,t}$ is the set of all channeling constraints. $\mathcal{P}^c = (k, \mathcal{X}, \mathcal{D}, \mathcal{C})$ is a combined model of m sub-models $\mathcal{P}_s$ for $1 \le s \le m$, where $k = \sum_{1 \le s \le m} k_s$, $\mathcal{X} = \bigcup_{1 \le s \le m} \mathcal{X}_s$, and $\mathcal{C} = (\bigcup_{1 \le s \le m} \mathcal{C}_s) \cup \mathcal{C}^c$. Function $f_{s,t}$ is a bijective channel function from assignments in $\mathcal{P}_s$ to those in $\mathcal{P}_t$. By definition, $f_{t,s} = f_{s,t}^{-1}$ and $f_{s,s}$ is the identity function. $\vartheta_t(x_{s,i}) = \{f_{s,t}(x_{s,i} \mapsto a) \,|\, a \in D_{x_{s,i}}\}$ is a set of all the corresponding assignments of $x_{s,i}$ in $\mathcal{P}_t$.

Definition 4.1 *Let $\mathcal{P}^c$ be a combined model of m sub-models $\mathcal{P}_s$ for $1 \le s \le m$.*

- *An assignment $x_{s,i} \mapsto a$ is m-*channeling node consistent* (m-NC$^*_\mathrm{c}$) if $C_\emptyset \oplus \sum_t C_{x_{t,j}}(b_t) < \top$, where $f_{s,t}(x_{s,i} \mapsto a) = x_{t,j} \mapsto b_t$ for $1 \le t \le m$.*
- *A variable $x_{s,i} \in \mathcal{X}$ is m-NC$^*_\mathrm{c}$ if (1) all assignments of $x_{s,i}$ are m-NC$^*_\mathrm{c}$ and (2) for $1 \le t \le m$, there exists an assignment $(x_{t,j} \mapsto b) \in \vartheta_t(x_{s,i})$ such that $C_{x_{t,j}}(b) = \bot$. The assignment $x_{t,j} \mapsto b$ is a* c-support *for $\vartheta_t(x_{s,i})$.*

- $\mathcal{P}^c$ *is* m-NC^*_c *if every variable in* $\mathcal{X}$ *is* m-NC^*_c.

Example 4.2 *Consider the combined model* $\mathcal{P}^c$ *in Figure 4(f). It is NC* but* not 2-NC^*_c, *since (1)* $C_\emptyset \oplus C_{x_1}(1) \oplus C_{y_1}(1) = 3 \oplus 3 \oplus 2 = \top$, *and (2) there are no c-supports for the tuple* $\theta = \{x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 1\}$. *Figure 4(h) shows an equivalent WCSP which is* 2-NC^*_c. ■

Note that 1-NC^*_c is equivalent to NC*, while m-NC^*_c achieves more prunings than NC*.

Following Debruyne and Bessiere [12], we define some notions to compare the strengths of two local consistencies. Suppose Φ_1 and Φ_2 are two local consistencies. Φ_1 is *stronger* [12] than Φ_2 if in any (W)CSP in which Φ_1 holds, Φ_2 holds too. Φ_1 is *strictly stronger* [12] than Φ_2 if Φ_1 is (1) stronger than Φ_2 and (2) there is at least one (W)CSP in which Φ_2 holds but Φ_1 does not hold.

Theorem 4.1 *Let* $\mathcal{P}^c$ *be a combined model of* m *sub-models* $\mathcal{P}_s$ *for* $1 \leq s \leq m$. m-NC^*_c *is strictly stronger than NC* on* $\mathcal{P}^c$.

Proof 4.1 *Without loss of generality, we give the proof for the case* $m = 2$ *and permutation WCSPs. The proof can be generalized to other classes of WCSPs.*

Let $\mathcal{P}^c$ *be a combined model consisting of two sub-models,* $\mathcal{P}_1 = (k_1, \mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X})$ *and* $\mathcal{P}_2 = (k_2, \mathcal{Y}, \mathcal{D}_\mathcal{Y}, \mathcal{C}_\mathcal{Y})$. *First, we prove* m-NC^*_c *is as strong as NC*. Suppose* $\mathcal{P}^c$ *is* 2-NC^*_c *but not NC*. In particular, we assume* $x_i \mapsto j$ *and* $y_j \mapsto i$ *are not NC*. We have to consider two cases. Then, either one of the following two cases must be true.*

1. $\exists x_i \mapsto j$ *in* $\mathcal{P}_1$ *such that* $C_\emptyset \oplus C_{x_i}(j) = \top = k_1 \oplus k_2$ *or* $\exists y_j \mapsto i$ *in* $\mathcal{P}_2$ *such that* $C_\emptyset \oplus C_{y_j}(i) = \top = k_1 \oplus k_2$.

   By the definition of 2-NC^*_c, *the assignments* $x_i \mapsto j$ *and* $y_j \mapsto i$ *cannot be removed if* $C_\emptyset \oplus C_{x_i}(j) \oplus C_{y_j}(i) < k_1 + k_2$. *Thus,* $C_\emptyset \oplus C_{x_i}(j) < k_1 + k_2$ *and* $C_\emptyset \oplus C_{y_j}(i) < k_1 + k_2$. *However, the assumption states that* $\mathcal{P}^c$ *is not NC* and there exists* $x_i \mapsto j$ *such that* $C_\emptyset \oplus C_{x_i}(j) = k_1 + k_2$. *This leads to a contradiction and this case cannot be true.*

2. $\forall a \in D_{x_i}$, $C_{x_i}(a) > \perp$ *or* $\forall b \in D_{y_j}$, $C_{y_j}(b) > \perp$.

   By the definition of 2-NC^*_c, *variable* x_i *in* $\mathcal{P}^c$ *and thus* $\mathcal{P}_1$ *has an assignment* $x_i \mapsto a$ *such that* $C_{x_i}(a) = \perp$. *This contradicts the assumption that the unary costs of the assignments of variable* x_i *are all greater than* $\perp$. *Thus, this case cannot be true also.*

Since both cases can never be true, m-NC^*_c *is as strong as NC*.*

Second, to show strictness, the WCSP in Figure 4(f) is NC but not* 2-NC^*_c. *Hence the result.* □

4.2.2 Enforcing m-NC$_c^*$

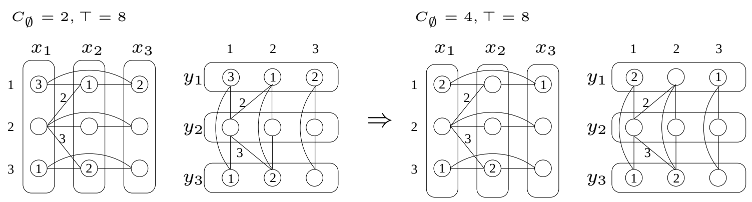
To enforce m-NC$_c^*$ on a combined model, we propose a new form of projection which can force *c-supports* for tuples.

Definition 4.2 *Given a tuple θ, let $\alpha = \min_{(x\mapsto a)\in\theta}\{C_x(a)\}$.* (Unary) c-projection *of a tuple θ over $C_\emptyset$ is a flow of α cost units such that $C_\emptyset := C_\emptyset \oplus \alpha$ and for each $(x \mapsto a) \in \theta$, $C_x(a) := C_x(a) \ominus \alpha$.*

C-projection is a generalization of ordinary projection. The former allows the assignments in θ to be from *different* variables, while the latter is equivalent to c-projection of *all* assignments of a *single* variable. Clearly, after c-projection of a tuple θ, there must exist an assignment $(x \mapsto a) \in \theta$ such that $C_x(a) = \perp$. Note that in a combined model, if θ corresponds to the set of *all assignments of one variable* in another sub-model, then c-projection of θ maintains the same cost distribution on complete tuples.

Theorem 4.2 *Let $x_{s,i}$ be a variable in a combined model $\mathcal{P}^c$. C-projection of $\vartheta_t(x_{s,i})$ over $C_\emptyset$ transforms $\mathcal{P}^c$ into an equivalent WCSP.*

Proof 4.2 *Let $x_{s,i}$ be a variable with domain $D_{x_{s,i}}$ in a combined model $\mathcal{P}^c$ with m sub-models. A solution of $\mathcal{P}^c$ must contain exactly one assignment among $\vartheta_t(x_{s,i})$. Therefore, there must be a necessary cost $\alpha = \min_{(x_{t,j}\mapsto b)\in\vartheta_t(x_{s,i})}\{C_{x_{t,j}}(b)\}$ incurred to any solution of $\mathcal{P}^c$. Also, only one of the unary costs among $\vartheta_t(x_{s,i})$ will be incurred in a solution of $\mathcal{P}^c$. Therefore, subtracting α from each of the unary costs among $\vartheta_t(x_{s,i})$ would only reduce the overall solution cost by α. By adding α to $C_\emptyset$, we compensate this reduction to the solution cost. Thus, c-projection of $\vartheta_t(x_{s,i})$ over $C_\emptyset$ transforms $\mathcal{P}^c$ into an equivalent WCSP.* □



(*a*) $\mathcal{P}^c$ before c-projection (*b*) $\mathcal{P}^c$ after c-projection

Figure 5: An example showing *c-projection* of $\vartheta_t(x_{s,i})$ over $C_\emptyset$

Example 4.3 *Consider the combined model $\mathcal{P}^c$ in Figure 5(a). None of the domain values of y_1 has a $\perp$ unary cost. Therefore, the minimum cost 1 among the set of assignments $\{y_1 \mapsto 1, y_1 \mapsto 2, y_1 \mapsto 3\}$ is projected over $C_\emptyset$. In the other sub-model, $\theta = \{x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 1\}$ corresponds to the set of all assignments of y_1. Hence, by Theorem 4.2, c-projection of θ over $C_\emptyset$ maintains the same cost distribution on complete tuples. In Figure 5(a), $C_{x_1}(1) = 3$, $C_{x_2}(1) = 1$, and $C_{x_3}(1) = 2$, c-projection of θ deducts 1 from each of these costs* *and increases $C_\emptyset$ by 1, forcing a c-support $x_2 \mapsto 1$ for θ. Figure 5(b) gives an equivalent WCSP after c-projection.* ■

Algorithm 4.1: Algorithm for enforcing m-NC$_\mathtt{c}^*$

1 **Function** $\mathtt{NC}_\mathtt{c}^*(C_\emptyset, k, \mathcal{X}, \mathcal{D}, \mathcal{C})$
2 **foreach** $(x_{s,i} \in \mathcal{X})$ **do**
3   **foreach** $(1 \leq t \leq m)$ **do**
4         $\alpha := \min_{(x_{t,j} \mapsto b) \in \vartheta_t(x_{s,i})}\{C_{x_{t,j}}(b)\}$;
5         $C_\emptyset := C_\emptyset \oplus \alpha$;
6   **foreach** $((x_{t,j} \mapsto b) \in \vartheta_t(x_{s,i}))$ **do**
7             $C_{x_{t,j}}(b) := C_{x_{t,j}}(b) \ominus \alpha$;
8 **foreach** $(x_{s,i} \in \mathcal{X})$ **do**
9     $\mathtt{PruneVar}_\mathtt{c}(x_{s,i})$;

10 **Function** $\mathtt{PruneVar}_\mathtt{c}(x_{s,i})$
11 changed := false;
12 **foreach** $(a \in D_{x_{s,i}})$ **do**
13   let $(x_{t,j} \mapsto b_t) = f_{s,t}(x_{s,i} \mapsto a)$ for $1 \leq t \leq m$;
14   **if** $(C_\emptyset \oplus \sum_t C_{x_{t,j}}(b_t) = \top)$ **then**
15   **foreach** $(1 \leq t \leq m)$ **do**
16             $D_{x_{t,j}} := D_{x_{t,j}} \setminus \{b_t\}$;
17   changed := true;
18 **return** changed;

Algorithm 4.1 shows how to enforce m-NC$_\mathtt{c}^*$ on a combined model $\mathcal{P}^c$. The algorithm first forces a c-support for each $(x_{t,j} \mapsto b) \in \vartheta_t(x_{s,i})$ by c-projecting each $\vartheta_t(x_{s,i})$ over $C_\emptyset$. Next, for each $x_{s,i} \in \mathcal{X}$, $\mathtt{PruneVar}_\mathtt{c}$ is called to prune any non-m-NC$_\mathtt{c}^*$ assignments. By using table lookup, a channel function can be implemented in $O(1)$ time. Therefore, $\mathtt{PruneVar}_\mathtt{c}$ and $\mathtt{NC}_\mathtt{c}^*$ runs in $O(md)$ and $O(mnd)$ time respectively, where d is the maximum domain size and $n = |\mathcal{X}|$.

Theorem 4.3 *Given a combined WCSP $\mathcal{P}^c$ with m sub-models, the $\mathtt{NC}_\mathtt{c}^*$ algorithm enforces m-channeling node consistency (m-NC$_\mathtt{c}^*$) on $\mathcal{P}^c$ and preserves all solutions of $\mathcal{P}^c$.*

Proof 4.3 *The proof makes use of the definition of m-NC$_\mathtt{c}^*$. In each sub-model $\mathcal{P}_t$, where $1 \leq t \leq m$, of the combined model $\mathcal{P}^c$, the $\mathtt{NC}_\mathtt{c}^*$ algorithm first forces c-supports for each $(x_{t,j} \mapsto b) \in \vartheta_t(x_{s,i})$ by projecting each $\vartheta_t(x_{s,i})$ over $C_\emptyset$ (lines 2–7). This ensures there must exist an assignment $(x_{t,j} \mapsto b) \in \vartheta_t(x_{s,i})$ that $C_{x_{t,j}}(b) = \bot$. Besides, $\mathtt{PruneVar}_\mathtt{c}$ checks if $C_\emptyset \oplus \sum_t C_{x_{t,j}}(b_t) < \top$, where $f_{s,t}(x_{s,i} \mapsto a) = x_{t,j} \mapsto b_t$ for $1 \leq t \leq m$. It removes all values from domains of variables that cannot be extended to solutions. Thus, the $\mathtt{NC}_\mathtt{c}^*$ algorithm enforces*

m-channeling node consistency (m-NC$_\mathrm{c}^*$) on $\mathcal{P}^c$ and preserves all solutions of $\mathcal{P}^c$. □

When enforcing m-NC$_\mathrm{c}^*$ on a combined model $\mathcal{P}^c$, whenever an assignment $x_{s,i} \mapsto a$ is detected not m-NC$_\mathrm{c}^*$, all the corresponding assignments $f_{s,t}(x_{s,i} \mapsto a)$ for $1 \le t \le m$ are also not m-NC$_\mathrm{c}^*$ and can be pruned. Therefore, enforcing m-NC$_\mathrm{c}^*$ on $\mathcal{P}^c$ has already entailed all the channeling constraints in $\mathcal{C}^c$, and for efficiency, we can skip the postings of the channeling constraints in $\mathcal{P}^c$ to save propagation overhead. In subsequent discussions, we assume that *there will be no channeling constraints in a combined model if m-NC$_\mathrm{c}^*$ is enforced*.

4.3 Arc Consistency Revisited

Besides NC*, we also investigate the adverse behavior encountered when enforcing AC* in combined models. We continue with the WCSPs in Figure 4.

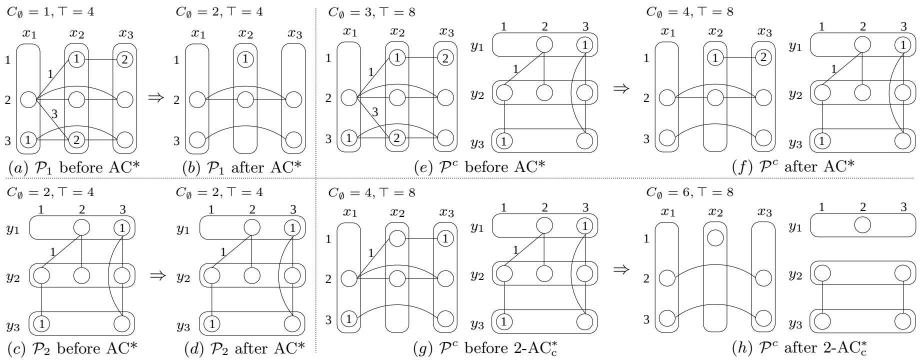


Figure 6: Enforcing arc consistencies on $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}^c$

Consider enforcing AC* on the WCSPs $\mathcal{P}_1$ and $\mathcal{P}_2$ in Figures 6(a) and 6(c) individually. In the former model, there are no supports for $x_1 \mapsto 2$ in C_{x_1,x_2}. Projection of C_{x_1,x_2} over C_{x_1} makes $C_{x_1}(2) = 1$, $C_{x_1,x_2}(2,1) = \perp$, $C_{x_1,x_2}(2,2) = \top$, and $C_{x_1,x_2}(2,3) = 2$. C_{x_1} loses its support and subsequently, C_{x_1} is projected over $C_\emptyset$ such that $C_\emptyset$ is increased to 2, $C_{x_1}(2) = \perp$, and $C_{x_1}(3) = \perp$. This leads to further removal of values $3 \in D_{x_2}$ and $1 \in D_{x_3}$ since $C_\emptyset \oplus C_{x_2}(3) = \top$ and $C_\emptyset \oplus C_{x_3}(1) = \top$, resulting in the WCSP in Figure 6(b). The latter model in Figure 6(c) is already AC*. Therefore, it remains the same after enforcing AC*.

Figure 6(e) shows the combined model $\mathcal{P}^c$ of those in Figures 6(a) and 6(c), and Figure 6(f) shows the result after enforcing AC* on $\mathcal{P}^c$. Figure 6(f) is obtained by (1) removing value $3 \in D_{x_2}$ due to the channeling constraints, (2) projection of C_{x_1,x_2} over C_{x_1}, and (3) projection of C_{x_1} over $C_\emptyset$.

4.3.1 Refining the Arc Consistency Definition

Consider variable x_2 in $\mathcal{P}_1$, the set of assignments $\{x_2 \mapsto 1, x_2 \mapsto 2, x_2 \mapsto 3\}$ in $\mathcal{P}_1$ corresponds to $\theta = \{y_1 \mapsto 2, y_2 \mapsto 2, y_3 \mapsto 2\}$ in $\mathcal{P}_2$. We have discussed in section 4.2.1 that a complete tuple of the combined model $\mathcal{P}^c$ must contain exactly one assignment among θ. Given an assignment $y_i \mapsto a$ in $\mathcal{P}_2$ but not in θ, there is a binary cost $C_{y_i,y_j}(a, 2)$ incurred between $y_i \mapsto a$ and $(y_j \mapsto 2) \in \theta$. (When $i = j$, there are actually no such binary costs, but we assume without loss of generality that the cost between two assignments of the same variable is $\top$.) Let $y_i \mapsto a$ be $y_2 \mapsto 1$ as an example. In Figure 6(f), since the minimum binary cost among $C_{y_2,y_1}(1, 2) = 1$ and "$C_{y_2,y_2}(1, 2)$" $= \top$ is $1 > \perp$, we can use such information to tighten the bound on the unary cost $C_{y_2}(1)$.

We capture this idea to propose a new arc consistency notion m-AC$^*_\mathrm{c}$ for combined models with m sub-models. The notion again is not restricted to permutation WCSPs only.

Definition 4.3 *Let $\mathcal{P}^c$ be a combined model of m sub-models $\mathcal{P}_s$ for $1 \leq s \leq m$.*

- *An assignment $x_{s,i} \mapsto a$ in $\mathcal{P}^c$ is m-*channeling arc consistent *(m-AC$^*_\mathrm{c}$) with respect to constraint $C_{x_{s,i},x_{s,j}}$ if for $1 \leq t \leq m$, there exists an assignment $(x_{t,j'} \mapsto b') \in \vartheta_t(x_{s,j})$ such that $C_{x_{t,i'},x_{t,j'}}(a', b') = \perp$, where $x_{t,i'} \mapsto a' = f_{s,t}(x_{s,i} \mapsto a)$. The assignment $x_{t,j'} \mapsto b'$ is a* c-support *for $x_{t,i'} \mapsto a'$.*
- *A variable $x_{s,i} \in \mathcal{X}$ is m-*AC$^*_\mathrm{c}$ *if all assignments of $x_{s,i}$ are m-AC$^*_\mathrm{c}$ with respect to all constraints involving $x_{s,i}$.*
- *$\mathcal{P}^c$ is m-*AC$^*_\mathrm{c}$ *if each $x_{s,i} \in \mathcal{X}$ is m-NC$^*_\mathrm{c}$ and m-AC$^*_\mathrm{c}$.*

Example 4.4 *Consider the combined model $\mathcal{P}^c$ in Figure 6(f). The WCSP is AC* but* not *2-AC$^*_\mathrm{c}$, since there are no c-supports among $\{y_1 \mapsto 2, y_2 \mapsto 2\}$ for $y_2 \mapsto 1$. Figure 6(h) shows an equivalent 2-AC$^*_\mathrm{c}$ WCSP. An optimal solution, $\{x_1 \mapsto 2, x_2 \mapsto 1, x_3 \mapsto 3, y_1 \mapsto 2, y_2 \mapsto 1, y_3 \mapsto 3\}$, has aggregate cost 6.* ■

Again, 1-AC$^*_\mathrm{c}$ is equivalent to AC*, while m-AC$^*_\mathrm{c}$ is a stronger notion of consistency than AC*.

Theorem 4.4 *Let $\mathcal{P}^c$ be a combined model of m sub-models $\mathcal{P}_s$ for $1 \leq s \leq m$. m-AC$^*_\mathrm{c}$ is strictly stronger than AC* on $\mathcal{P}^c$.*

Proof 4.4 *Without loss of generality, we prove the case $m = 2$ and permutation WCSPs. The proof can be generalized to other classes of WCSPs.*

Let $\mathcal{P}^c$ be a combined model consisting of two sub-models, $\mathcal{P}_1 = (k_1, \mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X})$ and $\mathcal{P}_2 = (k_2, \mathcal{Y}, \mathcal{D}_\mathcal{Y}, \mathcal{C}_\mathcal{Y})$. Suppose $\mathcal{P}^c$ is 2-AC$^*_\mathrm{c}$ but not AC. This means that there exists $a \in D_{x_i}$ such that for all the assignments $x_j \mapsto b$, $C_{x_i,x_j}(a, b) > \perp$. However, by the definition of 2-AC$^*_\mathrm{c}$, there exists an assignment $x_j \mapsto b$ such that $C_{x_i,x_j}(a, b) = \perp$. This leads to a contradiction and 2-AC$^*_\mathrm{c}$ is as strong as AC*.*

To show strictness, the combined model $\mathcal{P}^c$ in Figure 6(f) is AC but not 2-AC$^*_\mathrm{c}$. Hence the result.* □

4.3.2 Enforcing m-AC$_c^*$

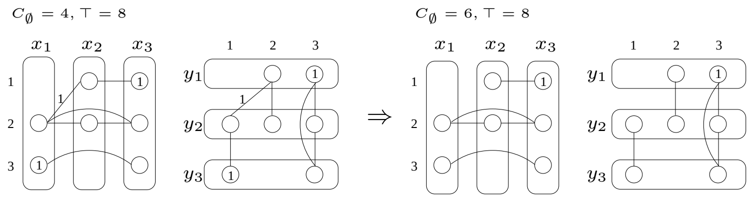
To enforce m-AC$_c^*$ on a combined model, we extend the definition of c-projections which can force c-supports for assignments.

Definition 4.4 *Given an assignment $x \mapsto a$ and a tuple θ where $(x \mapsto a) \notin \theta$, let $\alpha = \min_{(y \mapsto b) \in \theta}\{C_{x,y}(a,b)\}$.* (Binary) c-projection *of a tuple θ over $x \mapsto a$ is a flow of α cost units such that $C_x(a) := C_x(a) \oplus \alpha$ and for each $(y \mapsto b) \in \theta$, $C_{x,y}(a,b) := C_{x,y}(a,b) \ominus \alpha$.*

Binary c-projection of a tuple θ is equivalent to ordinary binary projection if θ is the set of *all* assignments of a *single* variable. In a combined model, if θ corresponds to the set of all assignments of a single variable in another sub-model, then c-projection of θ over an assignment not in θ yields an equivalent WCSP.

Theorem 4.5 *Let $x_{s,i}$ be a variable in a combined model $\mathcal{P}^c$. C-projection of $\vartheta_t(x_{s,i})$ over an assignment $x_{t,j} \mapsto a \notin \vartheta_t(x_{s,i})$ transforms $\mathcal{P}^c$ into an equivalent WCSP.*

Proof 4.5 *Let $x_{s,i}$ be a variable in a combined model $\mathcal{P}^c$ with m sub-models. Any solution of $\mathcal{P}^c$ must contain exactly one assignment among $\vartheta_t(s_{s,i})$. Suppose there is another assignment $x_{t,j} \mapsto a \notin \vartheta(x_{s,i})$, there must be a necessary binary cost $\beta = \min_{(x_{t,j'} \mapsto b) \in \vartheta_t(x_{s,i})}\{C_{x_{t,j},x_{t,j'}}(a,b)\}$ incurred between $x_{t,j} \mapsto a$ and all assignments in $\vartheta_t(x_{s,i})$. In c-projection, the minimum cost β is subtracted from each of the binary costs, but is compensated by adding β to the unary cost $C_{x_{t,j}}(a)$. Thus, the solution cost remains unchanged and c-projections of $\vartheta_t(x_{s,i})$ over the assignment $x_{t,j} \mapsto a \notin \vartheta(x_{s,i})$ transformed $\mathcal{P}^c$ into an equivalent WCSP.* $\square$



(a) $\mathcal{P}^c$ before enforcing 2-AC$_c^*$ (b) Intermediate step of 2-AC$_c^*$ on $\mathcal{P}^c$

Figure 7: An example showing *c-projection* $\vartheta_t(x_{s,i})$ over $x_{t,j} \mapsto a \notin \vartheta_t(x_{s,i})$

Example 4.5 *Figure 7(a) repeats the combined model $\mathcal{P}^c$ in Figure 6(g). The assignment $y_2 \mapsto 1$ has no c-supports in $\theta = \{y_1 \mapsto 2, y_2 \mapsto 2\}$, which is the set of assignments in $\mathcal{P}_2$ corresponding to the set of all assignments of x_2 in $\mathcal{P}_1$. Thus, c-projections of θ over $y_2 \mapsto 1$ yields $C_{y_2,y_1}(1,2) = \bot$ and $C_{y_2}(1) = 1$. Two sets of assignments $\{x_1 \mapsto 2, x_1 \mapsto 3\}$ and $\{y_2 \mapsto 1, y_3 \mapsto 1\}$ lose their c-supports and the minimum cost 1 from each set of assignments is projected over $C_\emptyset$, increasing*

$C_\emptyset$ from 4 to 6, as shown in Figure 7(b). The assignments $x_3 \mapsto 1$ and $y_1 \mapsto 3$ are consequently not 2-NC$^*_\mathrm{c}$, since $C_\emptyset \oplus C_{x_3}(1) \oplus C_{y_1}(3) = 6 \oplus 1 \oplus 1 = \top$, and are thus pruned. Variable y_1 is now bound and further propagation yields the final 2-AC$^*_\mathrm{c}$ WCSP in Figure 6(h). ■

Algorithm 4.2: Algorithm for enforcing m-AC$^*_\mathrm{c}$

Function $\texttt{AC}^*_\mathrm{c}(C_\emptyset, k, \mathcal{X}, \mathcal{D}, \mathcal{C})$
$\mathcal{Q} := \mathcal{X}$;
while $(\mathcal{Q} \neq \emptyset)$ **do**
  $x_{s,i} := \texttt{Pop}(\mathcal{Q})$;
  foreach $(C_{x_{s,i},x_{s,j}} \in \mathcal{C})$ **do**
    $\texttt{FindCSupport}(x_{s,i}, x_{s,j})$;
  foreach $(x_{s,i} \in \mathcal{X})$ **do**
    if $(\texttt{PruneVar}_\mathrm{c}(x_{s,i}))$ **then**
      $\mathcal{Q} := \mathcal{Q} \cup \{x_{s,i}\}$;

Function $\texttt{FindCSupport}(x_{s,i}, x_{s,j})$
foreach $(1 \leq t \leq m)$ **do**
  supported := true;
  foreach $((x_{t,i'} \mapsto a') \in \vartheta_t(x_{s,i}))$ **do**
    if $(S(x_{t,i'} \mapsto a', x_{s,j}, t) \notin \vartheta_t(x_{s,j}))$ **then**
      let $(x_{t,j^*} \mapsto b^*) = argmin_{(x_{t,j'} \mapsto b') \in \vartheta_t(x_{s,j})}\{C_{x_{t,i'},x_{t,j'}}(a', b')\}$;
      $S(x_{t,i'} \mapsto a', x_{s,j}, t) := (x_{t,j^*} \mapsto b^*)$;
      $\alpha := C_{x_{t,i'},x_{t,j^*}}(a', b^*)$;
      $C_{x_{t,i'}}(a') := C_{x_{t,i'}}(a') \oplus \alpha$;
      foreach $((x_{t,j'} \mapsto b') \in \vartheta_t(x_{t,j}))$ **do**
        $C_{x_{t,i'},x_{t,j'}}(a', b') := C_{x_{t,i'},x_{t,j'}}(a', b') \ominus \alpha$;
      if $(C_{x_{t,i'}}(a') = \bot$ *and* $\alpha > \bot)$ **then**
        supported := false;
  if $(\neg$supported$)$ **then**
    $\texttt{ProjectUnary}(x_{s,i})$;

Function $\texttt{ProjectUnary}(x_{s,i})$
let $(x_{t,i^*} \mapsto a^*) = argmin_{(x_{t,i'} \mapsto a') \in \vartheta_t(x_{s,i})}\{C_{x_{t,i'}}(a')\}$;
$S(x_{s,i}, t) := (x_{t,i^*} \mapsto a^*)$;
$\alpha := C_{x_{t,i^*}}(a^*)$;
$C_\emptyset := C_\emptyset \oplus \alpha$;
foreach $((x_{t,i'} \mapsto a') \in \vartheta_t(x_{s,i}))$ **do**
  $C_{x_{t,i'}}(a') := C_{x_{t,i'}}(a') \ominus \alpha$;

Algorithm 4.2 shows how to enforce m-AC$_\mathrm{c}^*$ on a combined model $\mathcal{P}^c$. It uses two data structures for storing c-supports. $S(x_{t,i'} \mapsto a', x_{s,j}, t)$ stores the current c-support for the assignment $x_{t,i'} \mapsto a'$ among $\vartheta_t(x_{s,j})$, while $S(x_{s,i}, t)$ stores the current c-support for $\vartheta_t(x_{s,i})$. `FindCSupport` generalizes `FindSupport` in the algorithm for enforcing AC* [26, 28] so that for each sub-model $\mathcal{P}_t$, it forces a c-support among $\vartheta_t(x_{s,j})$ for each assignment $(x_{t,i'} \mapsto a') \in \vartheta_t(x_{s,i})$. C-projections over $C_\emptyset$ is done by `ProjectUnary` when necessary. After finding c-supports, any assignments that are not m-NC$_\mathrm{c}^*$ are pruned using `PruneVar`$_\mathtt{c}$ in Algorithm 4.1. `FindCSupport` algorithm runs in $O(md^2)$ time, hence the overall m-AC$_\mathrm{c}^*$ algorithm runs in $O(mn^2d^3)$ time. Note that in practice, redundant modeling is usually done on combining two models, i.e., $m = 2$. Thus, the 2-AC$_\mathrm{c}^*$ algorithm is only a *constant* factor worse than the traditional AC* algorithm.

Theorem 4.6 *Given a combined WCSP $\mathcal{P}^c$ with m sub-models, the m-AC$_\mathrm{c}^*$ algorithm enforces m-channeling arc consistency (m-AC$_\mathrm{c}^*$) on $\mathcal{P}^c$ and preserves all solutions of $\mathcal{P}^c$.*

Proof 4.6 *In each sub-model $\mathcal{P}_t$, where $1 \leq t \leq m$, of the combined model $\mathcal{P}^c$, the m-AC$_\mathrm{c}^*$ algorithm first forces c-supports for each assignment $(x_{t,i'} \mapsto a') \in \vartheta_t(x_{s,i})$ by projecting each $\vartheta_t(x_{s,i})$ over $C_{x_{t,i'}}(a')$. This ensures there must exist an assignment $(x_{t,j'} \mapsto b') \in \vartheta_t(x_{s,i})$ that $C_{x_{t,i'},x_{t,j'}}(a', b') = \perp$.* `ProjectUnary` *enforces m-channeling node consistency m-NC$_\mathrm{c}^*$ on $\mathcal{P}^c$ if the variables lose its consistencies when forcing supports for m-AC$_\mathrm{c}^*$. Besides,* `PruneVar`$_\mathtt{c}$ *checks if $C_\emptyset \oplus \sum_t C_{x_{t,j}}(b_t) < \top$, where $f_{s,t}(x_{s,i} \mapsto a) = x_{t,j} \mapsto b_t$ for $1 \leq t \leq m$. It removes all values from domains of variables that cannot be extended to solutions. Besides, the m-AC$_\mathrm{c}^*$ algorithm has a queue $\mathcal{Q}$ for storing the variables whose domains have been changed. $\mathcal{Q}$ is initialized to contain all the variables in $\mathcal{X}$ because every variable has to obtain an initial support for its values with respect to every binary constraints. The algorithm stops when $\mathcal{Q}$ becomes empty, indicating every variables in $\mathcal{P}^c$ is m-AC$_\mathrm{c}^*$. Thus, the m-AC$_\mathrm{c}^*$ algorithm enforces m-channeling node consistency (m-AC$_\mathrm{c}^*$) on $\mathcal{P}^c$ and preserves all solutions of $\mathcal{P}^c$.* □

5 A Parameterized Local Consistency

In previous section, we proposed to refine the definitions of NC* and AC* for combined models. Sometimes, one may want to use other local consistencies to solve a problem, making this approach inapplicable. Thus, in this section, we describe our second approach, which is applicable to any local consistency Φ. A parameterized local consistency LB(m,Φ), which maintains the same local consistency Φ on each sub-model, and its enforcement algorithm are proposed for the suggested combined model. Theoretical comparisons are then made between the proposed consistency and an existing approach for redundant modeling. The semantics of the three approaches and the relation between the combined models and their sub-models are also discussed.

5.1 Combining Mutually Redundant Models

Consider m mutually redundant models $\mathcal{P}_i = (k, \mathcal{X}_i, \mathcal{D}_i, \mathcal{C}_i)$ for $1 \le i \le m$ of the same problem. Here, we proposed another approach to connect them. This combined model has two major differences to the one used in Section 4.

First, the value k remains unchanged for each sub-model $\mathcal{P}_i$ in the combined model $\mathcal{P}^c$, instead of summing the values k of all $\mathcal{P}_i$ as the value k of $\mathcal{P}^c$. Second, we associate each sub-model $\mathcal{P}_i$ with a *local zero-arity constraint* $C^i_\emptyset$ to denote the local lower bound of costs in $\mathcal{P}_i$. Now, enforcing NC* sends the costs of the unary constraints in $\mathcal{P}_i$ to $C^i_\emptyset$ as each $\mathcal{P}_i$ has its own $C^i_\emptyset$. Thus, the global lower bound of costs $C_\emptyset$ of the combined model $\mathcal{P}^c$ takes the maximum value of all local lower bounds $C^i_\emptyset$ of $\mathcal{P}_i$ (i.e., $C_\emptyset = \max_i\{C^i_\emptyset\}$).

Based on the combined model, we denote a complete tuple in a sub-model $\mathcal{P}_i$ as a *semi-complete tuple* θ_i. The cost of every semi-complete tuple θ_i in $\mathcal{P}_i$ is the same and equals the cost of the complete tuple θ in $\mathcal{P}^c$. (i.e., $\mathcal{V}(\theta_i) = \mathcal{V}(\theta_j) = \mathcal{V}(\theta)$, for $1 \le i, j \le m$ and $i \ne j$).

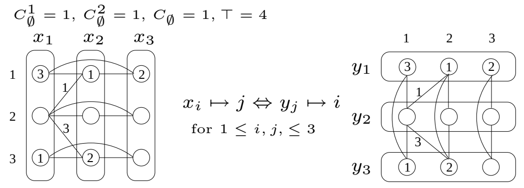


Figure 8: An example of a combined permutation WCSP model using second approach

Example 5.1 *Figure 8 shows the combined model $\mathcal{P}^c$ of $\mathcal{P}_1$ and $\mathcal{P}_2$ in Figure 2 using our second approach. In $\mathcal{P}^c$, the valuation structure remains $S(4)$ which is the same as that of $\mathcal{P}_1$ and $\mathcal{P}_2$, and there are six variables $\{x_1, x_2, x_3, y_1, y_2, y_3\}$. Each sub-model has its own local zero-arity constraints $C^1_\emptyset$ and $C^2_\emptyset$. $C_\emptyset$ of $\mathcal{P}^c$ takes the maximum of $C^1_\emptyset$ and $C^2_\emptyset$ (i.e., $C_\emptyset = \max\{C^1_\emptyset, C^2_\emptyset\} = \max\{1, 1\} = 1$).*
■

5.2 Enforcing Consistency on Combined Models

Given a combined model $\mathcal{P}^c$ with m mutually redundant sub-models $\mathcal{P}_i$. We can enforce local consistency Φ on each sub-model $\mathcal{P}_i$ of $\mathcal{P}^c$ and use the bijective channel function $f(x \mapsto a) = (y \mapsto b)$, defined in Section 3.2, to transmit instantiation and pruning information between $\mathcal{P}_i$ to ensure that the bijective mapping between assignments of any two sub-models $\mathcal{P}_i$ and $\mathcal{P}_j$ for $1 \le i < j \le m$ is maintained. Based on these ideas, we proposed a parameterized local consistency LB(m,Φ) for combined models $\mathcal{P}^c$ with m mutually redundant sub-models $\mathcal{P}_i$. Note that Φ can be any local consistency that can be applied to a single WCSP model.

Definition 5.1 *Let $\mathcal{P}^c$ be a combined model of m mutually redundant sub-models $\mathcal{P}_s$ for $1 \leq s \leq m$, Φ be a local consistency, and $f_{s,t}$ be a bijective channel function from assignments of $\mathcal{P}_s$ to assignments of $\mathcal{P}_t$ for all $1 \leq s < t \leq m$. $\mathcal{P}^c$ is said to be* LB(m,Φ) *if:*

1. *all sub-models $\mathcal{P}_s$ are Φ, and*
2. *for all assignments $x_{s,i} \mapsto a$ of $\mathcal{P}_s$, $a \in D_{x_{s,i}} \Leftrightarrow b \in D_{x_{t,j}}$, where $f_{s,t}(x_{s,i} \mapsto a) = x_{t,j} \mapsto b$.*

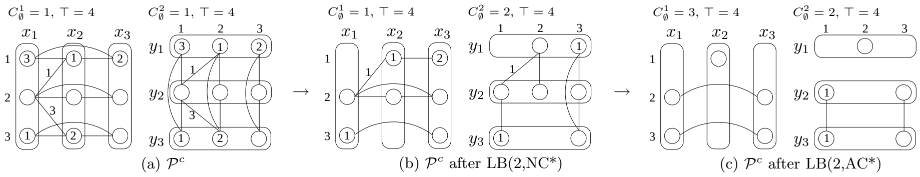


Figure 9: Enforcing node and arc consistencies on WCSPs $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}^c$

Example 5.2 *Consider the combined WCSP model $\mathcal{P}^c$ in Figure 9(a). It is not LB(2,NC*) since both sub-models $\mathcal{P}_1$ and $\mathcal{P}_2$ are not NC* (i.e., $C^1_\emptyset \oplus C_{x_1}(1) = 1 \oplus 3 = \top$, $C^2_\emptyset \oplus C_{y_1}(1) = 1 \oplus 3 = \top$.) After enforcing NC* on each sub-model, $1 \in x_1$, $1 \in y_1$, and $2 \in y_3$ are pruned. By sharing the pruning information between sub-models, $3 \in x_2$ is also pruned. Figure 9(b) gives an equivalent combined WCSP model $\mathcal{P}^c$, which is now LB(2,NC*). However, it is still not LB(2,AC*) since sub-models $\mathcal{P}_1$ is not AC*. Enforcing AC* on each sub-model and sharing the pruning information between sub-models yield an equivalent combined WCSP model $\mathcal{P}^c$ in Figure 9(c), which is now LB(2,AC*).* ■

Algorithm 5.1: Algorithms for enforcing LB(m,Φ)

```
1  Function LB(m, Φ, P^c)
2  repeat
3      foreach sub-model P_s of P^c do
4          enforce Φ on P_s;
5      foreach pair of sub-models P_s, P_t of P^c (s ≠ t) do
6          foreach x_{s,i} ∈ X_s do
7              foreach a ∈ D_{x_{s,i}} do
8                  if b ∉ D_{x_{t,j}} where x_{t,j} ↦ b = f_{s,t}(x_{s,i} ↦ a) then
9                      remove a from D_{x_{s,i}};
10 until P^c remains unchanged ;
```

LB(m,Φ) can be enforced using a simple algorithm shown in Algorithm 5.1. We first enforce Φ on each sub-model (lines 3–4). This ensures that all sub-models $\mathcal{P}_s$ satisfy the Φ property (condition 1). For condition 2, if there is a value $a \in D_{x_{s,i}}$ being pruned in one sub-model $\mathcal{P}_s$, the corresponding value $b \in D_{x_{t,j}}$ obtained via the channel function will also be pruned in other sub-models $\mathcal{P}_t$ for $1 \leq t \leq m$ and $s \neq t$ (lines 5–9). The process repeats until there are no more changes in any sub-models, and $\mathcal{P}^c$ is then LB(m,Φ). Since each sub-model $\mathcal{P}_s$ has its local lower bound $C^s_\emptyset$, unary constraints are projected towards its own $C^s_\emptyset$ when enforcing NC*. For example, unary constraints in $\mathcal{P}_1$ are projected to $C^1_\emptyset$ and those in $\mathcal{P}_2$ are projected to $C^2_\emptyset$. During constraint propagation, when the global lower bound $C_\emptyset$ of the combined model $\mathcal{P}^c$ reaches the global upper bound $\top$ (i.e., $C_\emptyset = \max_s\{C^s_\emptyset\} = \top$), this means that there exists at least one sub-model $\mathcal{P}_s$ in which its local lower bound $C^s_\emptyset$ is increased to $\top$, and we cannot extend a tuple of this sub-model to obtain a solution. Since all the sub-models are mutually redundant to each other, the other sub-models will eventually lead to failure. Therefore, a backtrack is triggered in the search. The following theorem states that the algorithm in Algorithm 5.1 enforces LB(m,Φ).

Theorem 5.1 *Let $\mathcal{P}^c$ be a combined model of m WCSP sub-models $\mathcal{P}_s$ for $1 \leq s \leq m$, and Φ be any local consistency. The LB($m, \Phi, \mathcal{P}^c$) algorithm transforms $\mathcal{P}^c$ into an equivalent combined model $\mathcal{P}^{c\prime}$.*

Proof 5.1 *When a combined model $\mathcal{P}^c$ is passed to the LB($m, \Phi, \mathcal{P}^c$) algorithm, enforcing Φ on each sub-model $\mathcal{P}_i$ transforms $\mathcal{P}_i$ to an equivalent sub-model $\mathcal{P}'_i$. Furthermore, the mutual redundancy of two sub-models $\mathcal{P}_s$ and $\mathcal{P}_t$ guarantees that if value b is not in $D_{x_{t,j}}$, then value a must not be in the domain of $x_{s,i}$, where $x_{t,j} \mapsto b = f_{s,t}(x_{s,i} \mapsto a)$. Thus, removing value a from $D_{x_{s,i}}$ in line 9 will not remove any values that belong to a solution of the combined model $\mathcal{P}^c$. Hence, upon termination of the algorithm, the transformed model $\mathcal{P}^{c\prime}$ is equivalent to the input model $\mathcal{P}^c$.* □

Furthermore, LB(m,Φ) is a stronger notion of consistency than Φ.

Theorem 5.2 *Let $\mathcal{P}^c$ be a combined model of m sub-models $\mathcal{P}_s$ for $1 \leq s \leq m$. Enforcing LB(m,Φ) on $\mathcal{P}^c$ is strictly stronger than enforcing Φ on any $\mathcal{P}_s$.*

Proof 5.2 *By definition 5.1, $\mathcal{P}^c$ is LB(m,Φ) if all sub-models $\mathcal{P}_s$ for $1 \leq s \leq m$ are Φ. This means that LB(m,Φ) is stronger than Φ. To show strictness, consider the models $\mathcal{P}_1$ and $\mathcal{P}_2$ in Figures 4(a) and 4(c). The equivalent models in Figures 4(b) and 4(d) are NC*, but the combined model of these two sub-models is not LB(2,NC*) because $2 \notin D_{y_3}$ and $3 \in D_{x_2}$. Similarly, the models $\mathcal{P}_1$ and $\mathcal{P}_2$ in Figures 6(b) and 6(d) are AC*, but the combined model of these two sub-models is not LB(2,AC*) because $1 \notin D_{x_3}$ and $3 \in D_{y_1}$ The same result can be applied to other local consistencies Φ.* □

Note that unlike classical CSPs, enforcing AC* on a WCSP can result in more than one possible outcome, depending on the order of the domain values to be pruned and the constraints to be handled in an algorithm [28]. Therefore, although we have the "strictly stronger" notion, when comparing a LB(m,AC*) combined model and a AC* sub-model, we cannot guarantee that the domain of a variable in the combined model must be a subset of that in the single sub-model. Nonetheless, such theoretical comparison is still worthwhile as it shows that enforcing one local consistency can generally prune more domain values than enforcing another.

5.3 Theoretical Comparison Between the Two Approaches

Given m mutually redundant WCSPs $\mathcal{P}_s = (k_s, \mathcal{X}_s, \mathcal{D}_{\mathcal{X}s}, \mathcal{C}_s)$ for $1 \leq s \leq m$. In our first approach, the combined model $\mathcal{P}^c$ takes the sum of all values k_s of its sub-models $\mathcal{P}_s$ and has a top value which is the sum of those in $\mathcal{P}_s$. However, in our second approach, the combined model $\mathcal{P}^{c'}$ takes the same top value as each sub-model has. For example, Figure 4(g) gives a combined model $\mathcal{P}^c$ of two mutually redundant models $\mathcal{P}_1$ and $\mathcal{P}_2$ in Figures 4(a) and 4(c) respectively, with $C_\emptyset = 1 \oplus 1 = 2$ and $\top = 4 \oplus 4 = 8$. Besides, $\mathcal{P}^c$ has only one lower bound $C_\emptyset$; there are no individual lower bound for sub-models. Any cost that is sent from the unary constraints in any sub-model all goes to $C_\emptyset$. Since $C_\emptyset$ and $\top$ are shared among sub-models in $\mathcal{P}^c$, the local consistency has to be refined for combined models. Based on the combined model, we proposed new notions of node consistency m-NC$^*_\mathrm{c}$ and arc consistency m-AC$^*_\mathrm{c}$ to transmit pruning and cost movement information between sub-models. In our second approach, sub-models have their own local lower bounds; pruning information are transmitted via the channel function. Thus, modifications of local consistencies are not required.

When enforcing m-NC$^*_\mathrm{c}$ and m-AC$^*_\mathrm{c}$, not only the instantiation and pruning information but also the cost projection information is transmitted between sub-models in the combined model. Transmitting cost projection information can further discover and remove more node inconsistent values or increase the global lower bound. Thus, enforcing 2-NC$^*_\mathrm{c}$ and 2-AC$^*_\mathrm{c}$ achieves more constraint propagation than enforcing LB(2,NC*) and LB(2,AC*) respectively.

Theorem 5.3 *Let $\mathcal{P}^c$ and $\mathcal{P}^{c'}$ be combined models of m mutually redundant sub-models $\mathcal{P}_s$ for $1 \leq s \leq m$ formed using our first and second approaches respectively. Enforcing m-NC$^*_\mathrm{c}$ (*resp.* m-AC$^*_\mathrm{c}$) on $\mathcal{P}^c$ is* strictly stronger *than enforcing LB(m,NC*) (*resp.* LB(m,AC*)) on $\mathcal{P}^{c'}$.*

Proof 5.3 *Without loss of generality, we prove the case $m = 2$. The proof can be generalized to different numbers of sub-models.*

Let $\mathcal{P}^c$ and $\mathcal{P}^{c'}$ be the combined models, formed by our first and second approach respectively, consisting of two sub-models, $\mathcal{P}_1 = (k, \mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X})$ and $\mathcal{P}_2 = (k, \mathcal{Y}, \mathcal{D}_\mathcal{Y}, \mathcal{C}_\mathcal{Y})$. Suppose $\mathcal{P}^c$ is 2-NC$^*_\mathrm{c}$ but $\mathcal{P}^{c'}$ is not LB(2,NC). In particular, we assume $x_i \mapsto j$ and $y_j \mapsto i$ are not LB(2,NC*). First, we prove 2-NC$^*_\mathrm{c}$ is as strong as LB(2,NC*). We have to consider two cases: (1) By the definition of 2-NC$^*_\mathrm{c}$, the assignments $x_i \mapsto j$ and $y_j \mapsto i$ cannot be removed if $C_\emptyset \oplus C_{x_i}(j) \oplus C_{y_j}(i) < 2k$. However, the assumption states that $\mathcal{P}^{c'}$ is not LB(2,NC*) and there exists $x_i \mapsto j$ and $y_j \mapsto i$ such that $C^1_\emptyset \oplus C_{x_i}(j) = k$ or $C^2_\emptyset \oplus C_{y_j}(i) = k$ (i.e., $C^1_\emptyset \oplus C_{x_i}(j) \oplus C_{y_j}(i) = 2k$). This leads to a contradiction.*

(2) By the definition of 2-NC^*_{c}*, a variable* x_i *has an assignment* $x_i \mapsto a$ *such that* $C_{x_i}(a) = \bot$*. This contradicts the assumption that the unary costs of the assignments of variable* x_i *are all greater than* $\bot$*.*

To show strictness, consider the problem in Figure 9. Figure 9(b) gives the combined WCSP $\mathcal{P}^{c'}$ *which is LB(2,NC*). However, its corresponding combined WCSP* $\mathcal{P}^c$ *is not* 2-NC^*_{c} *because the global lower bound can be further increased from* 3 *(i.e.,* $C^1_\emptyset \oplus C^2_\emptyset = 1 \oplus 2 = 3$*) to* 4 *by subtracting unary cost* 1 *from both* $C_{x_2}(1)$ *and* $C_{x_3}(1)$*. Similarly, Figure 9(c) gives the combined WCSP* $\mathcal{P}^{c'}$ *which is LB(2,AC*). However, its corresponding combined WCSP* $\mathcal{P}^c$ *is not* 2-AC^*_{c} *because the global lower bound can be further increased from* 5 *(i.e.,* $C^1_\emptyset \oplus C^2_\emptyset = 3 \oplus 2 = 5$*) to* 6 *by subtracting unary cost* 1 *from both* $C_{y_2}(1)$ *and* $C_{y_3}(1)$*.* □

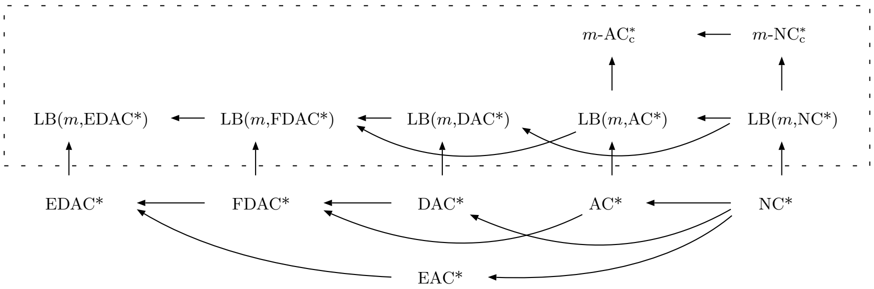


Figure 10: Relation between different local consistencies for WCSP

The relationship between different local consistencies for WCSP, including the proposed ones (placed inside the dashed rectangle) for combined WCSP models, is shown in Figure 10. An arrow $\Phi_1 \leftarrow \Phi_2$ indicates that Φ_1 is strictly stronger than Φ_2. There is no path from one local consistency to another if they are incomparable in terms of propagation strength. For the relationships between m-NC$^*_{\mathrm{c}}$ (*resp.* m-AC$^*_{\mathrm{c}}$) and other consistencies like FDAC* and EDAC*, further studies are needed on their propagation behaviors.

5.4 On Semantics of the Approaches

In this section, we explain the semantics of the naive, the first, and the second approaches and the relation between the combined models and their sub-models.

In the naive and the first approaches, a combined model is formed by connecting the sub-models using channeling constraints with a new k value and global lower bound $C_\emptyset$. We can always find a bijective mapping between the solutions in the original sub-models and the combined model.

Theorem 5.4 *Let* $\mathcal{P}^c$ *be a combined model of* m *WCSP sub-models* $\mathcal{P}_i$ *for* $1 \le i \le m$ *using the naive approach (*resp. *the first approach). For each complete assignment* θ_i *of the original* $\mathcal{P}_i$ *and a corresponding complete assignment* θ *of* $\mathcal{P}^c$*,* θ_i *is a solution in* $\mathcal{P}_i$ *if and only if* θ *is a solution in* $\mathcal{P}^c$*. The assignment* θ_i *equals the projection of* θ *over* $var(\theta_i)$*, and* $\mathcal{V}(\theta) = m \cdot \mathcal{V}(\theta_i)$*.*

Proof 5.4 *Without loss of generality, we give the proof for the case $m = 2$ and permutation WCSPs. The proof can be generalized to other classes of WCSPs. Let $\mathcal{P}^c$ be a combined model consisting of two sub-models $\mathcal{P}_1 = (k_1, \mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X})$ and $\mathcal{P}_2 = (k_2, \mathcal{Y}, \mathcal{D}_\mathcal{Y}, \mathcal{C}_\mathcal{Y})$.*

($\Rightarrow$) *Suppose θ_1 is a solution in $\mathcal{P}_1$, θ_2 is a solution in $\mathcal{P}_2$ via the channeling constraints, and the corresponding θ is not a solution in $\mathcal{P}^c$. According to the naive approach (*resp. *the first approach), the variables in $\mathcal{P}^c$ are the union of those in $\mathcal{P}_1$ and $\mathcal{P}_2$ (i.e., $\mathcal{X} \cup \mathcal{Y}$). Thus, $\theta = \theta_1 \cup \theta_2$. This leads to a contradiction.*

($\Leftarrow$) *Suppose θ is a solution in $\mathcal{P}^c$ and the corresponding θ_1 is not a solution $\mathcal{P}_1$. According to the naive approach (*resp. *the first approach), the variables in $\mathcal{P}^c$ are the union of those in $\mathcal{P}_1$ and $\mathcal{P}_2$ (i.e., $\mathcal{X} \cup \mathcal{Y}$). Thus, $\theta_1 \in \theta$. This leads to a contradiction.*

Since both cases cannot be true, hence θ_i is a solution in $\mathcal{P}_i$ if and only if θ is a solution in $\mathcal{P}^c$.

*According to the naive approach (*resp. *the first approach), the variables in $\mathcal{P}^c$ are the union of those in $\mathcal{P}_1$ and $\mathcal{P}_2$ (i.e., $\mathcal{X} \cup \mathcal{Y}$) and $\theta_1, \theta_2 \in \theta$. Thus, we can obtain θ_1 and θ_2 simply by the projection of θ over $\mathcal{X}$ and $\mathcal{Y}$ respectively.*

$\mathcal{P}_1$ and $\mathcal{P}_2$ are redundant models, they have the same cost distribution between every pair of equivalent solutions (i.e., $\mathcal{V}(\theta_1) = \mathcal{V}(\theta_2)$). In $\mathcal{P}^c$, all the unary costs are projected over the same $C_\emptyset$ of $\mathcal{P}^c$, the cost of the solution $\mathcal{V}(\theta)$ of $\mathcal{P}^c$ is thus 2 times the costs of solutions $\mathcal{V}(\theta_i)$ of any $\mathcal{P}_i$.

Hence the results. □

In the second approach, each sub-model $\mathcal{P}_i$ has their own $C^i_\emptyset$ to receive the projection of its unary costs. Thus, the combined model $\mathcal{P}^c$ is a new kind of WCSP. It has m local lower bounds $C^i_\emptyset$ of $\mathcal{P}_i$, and one global lower bound of costs $C_\emptyset$ of $\mathcal{P}^c$ which takes the maximum value of all $C^i_\emptyset$. Each sub-model $\mathcal{P}_i$ maintains its own local consistencies but share the assignment and pruning information among $\mathcal{P}_i$.

Theorem 5.5 *In the second approach, $\mathcal{P}^c$ is a new kind of combined WCSP model of m sub-models $\mathcal{P}_i$ for $1 \leq i \leq m$ in which every sub-model has its own local lower bound $C^i_\emptyset$ and the global lower bound of $\mathcal{P}^c$, $C_\emptyset = \max_i\{C^i_\emptyset\}$. For each complete assignment θ_i of the original $\mathcal{P}_i$ and a corresponding complete assignment θ of $\mathcal{P}^c$, θ_i is a solution in $\mathcal{P}_i$ if and only if θ is a solution in $\mathcal{P}^c$, and $\mathcal{V}(\theta) = \mathcal{V}(\theta_i)$.*

Proof 5.5 *Unlike the naive and first approach, the unary costs of $\mathcal{P}_i$ are projected over their own local lower bound $C^i_\emptyset$ and the global lower bound is determined by maximizing all $C^i_\emptyset$ (i.e., $C_\emptyset = \max_i\{C^i_\emptyset\}$). Thus, the cost of the corresponding solution $\mathcal{V}(\theta_i)$ for every sub-models $\mathcal{P}_i$ are the same.*

Similar to the proof of Theorem 5.4, there is a bijective mapping between the solutions of the original sub-model $\mathcal{P}_i$ and the combined model $\mathcal{P}^c$.

By the definition of the combined model $\mathcal{P}^c$ using the second approach, the cost of every θ_i in $\mathcal{P}_i$ is the same and equals the cost of θ in $\mathcal{P}^c$ (i.e., $\mathcal{V}(\theta) = \mathcal{V}(\theta_i)$).

Hence the results. □

6 Experiments

To evaluate the effectiveness and efficiency of the combined models, we implement our two proposed approaches in ToolBar [4], a branch and bound WCSP solver maintaining local consistencies at each search tree node. To strike a balance between the amount of information exchange and the overhead of extra models, we use only two sub-models in a combined model. Four benchmarks, knight's tour problem, Langford's problem, Latin square problem, and n-queens problem, which contain mainly binary constraints, are modeled as permutation WCSPs to test our approaches. Since naively enforcing AC* on combined models is not efficient (as shown in Table 1), comparisons are made directly among AC* [26], FDAC* [27], and EDAC* [11] on a single model $\mathcal{P}$, 2-AC^*_c on a combined model $\mathcal{P}^c$ proposed in Section 4, and LB(2,AC*), LB(2,FDAC*), and LB(2,EDAC*) on a combined model $\mathcal{P}^c$ proposed in Section 5. All combined models $\mathcal{P}^c$ contain a single model $\mathcal{P}$ and its induced model $\mathcal{P}'$, generated automatically using generalized model induction described in Section 3, as sub-models.

The experiments are run on a Sun Blade 2500 (2 $\times$ 1.6GHz US-IIIi) workstation with 2GB memory. We use the *dom/deg* variable ordering heuristic which chooses the variable with the smallest ratio of domain size to future degree. Values are chosen using the *smallest-cost-first* heuristic on $\mathcal{P}$ and the *smallest-cost-first* and the *dual-smallest-domain-first* [5] heuristics on $\mathcal{P}^c$. The dual-smallest-domain-first value ordering heuristic chooses the value whose corresponding variable in the other sub-model has the smallest domain size. The initial $\top$ value provided to the solver is n^2, where n is the number of variables in a model. Ten random models are generated for each soft instance and we report the average number of fails (i.e., the number of backtracks occurred in solving a model) and CPU time in seconds to find the first optimal solution for each instance. In the tables, the first column shows the problem instances; those marked with "*" are satisfiable and have a $\perp$ optimal cost. When solving these instances, once such solution is found, we need not prove its optimality and can terminate immediately. This is different from the case where the optimal solution has a non-$\perp$ cost, in which we still need to continue search to prove optimality. The subsequent columns show the results of enforcing various local consistencies on either $\mathcal{P}$ or $\mathcal{P}^c$. The best number of fails and CPU time among the results for each instance are highlighted in bold. A cell labeled with "-" denotes a timeout after two hours.

In the following, we briefly describe the four benchmark problems. Experimental results are given and compared between using existing algorithms and our two proposed approaches. Since WCSPs can degenerate to classical CSPs when the cost is either 0 or 1, we want to verify if our work on redundant modeling can benefit the same as shown by Cheng *et al.* [5] on classical CSPs. Thus, we report the results on classical instances as well for each benchmark.

Table 2: Experimental results on solving classical knight's tour problem using *smallest-cost-first* value ordering heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
(m,n)	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
(3,5)	672	0.07	672	0.07	672	0.10	**210**	0.09	**210**	0.08	**210**	**0.06**	**210**	0.07
(3,6)	4416	0.45	4416	0.53	4416	0.70	**816**	0.36	**816**	0.35	**816**	**0.29**	**816**	0.32
(3,7)*	545	**0.06**	545	0.07	545	0.10	**158**	0.08	**158**	0.08	**158**	0.08	**158**	0.08
(3,8)*	2657	0.33	2657	0.38	2657	0.50	**732**	0.34	**732**	0.33	**732**	**0.29**	**732**	0.33
(3,9)*	11535	1.66	11535	1.94	11535	2.61	**2005**	1.27	**2005**	1.21	**2005**	**1.06**	**2005**	1.20
(3,10)*	72183	11.79	72183	13.74	72183	18.72	**10628**	8.38	**10628**	7.85	**10628**	**6.79**	**10628**	7.59
(3,11)*	13225	2.17	13225	2.46	13225	3.39	**1878**	1.78	**1878**	1.70	**1878**	**1.32**	**1878**	1.54
(3,12)*	2349445	467.80	2349445	550.14	2349445	743.09	**212657**	268.81	**212657**	242.45	**212657**	**203.28**	**212657**	236.25
(3,13)*	766731	172.10	766731	202.52	766731	274.80	**49684**	96.56	**49684**	81.07	**49684**	**70.56**	**49684**	73.12
(3,14)*	-	-	-	-	-	-	-	-	**3679309**	7056.15	**3679309**	**5739.15**	**3679309**	6456.26
(3,15)*	1498214	367.45	1498214	426.99	1498214	588.97	**58889**	143.55	**58889**	136.88	**58889**	**109.21**	**58889**	111.14
(4,5)*	336	**0.03**	336	**0.03**	336	0.04	**77**	0.04	**77**	0.04	**77**	**0.03**	**77**	0.04
(4,6)*	7914	0.68	7914	0.76	7914	1.04	**1415**	0.55	**1415**	0.54	**1415**	**0.46**	**1415**	0.50
(4,7)*	40344	4.19	40344	4.62	40344	6.55	**5526**	2.67	**5526**	2.53	**5526**	**2.05**	**5526**	2.29
(4,8)*	1183823	145.18	1183823	161.48	1183823	228.35	**76661**	52.06	**76661**	49.32	**76661**	**39.91**	**76661**	43.68
(4,9)*	462506	67.28	462506	74.85	462506	105.82	**33575**	30.90	**33575**	27.64	**33575**	**22.99**	**33575**	25.21
(4,10)*	29317520	5166.23	29317520	5769.60	-	-	**971162**	1455.19	**971162**	1266.22	**971162**	**1026.10**	**971162**	1095.14

6.1 Knight's Tour Problem

The knight's tour problem is to find a sequence of moves by a knight on an $m \times n$ chessboard so that each square on the chessboard is traversed exactly once. We model this problem into a WCSP $\mathcal{P}$ using mn variables $\mathcal{X} = \{x_1, \ldots, x_{mn}\}$. Each variable x_i denotes a square on the chessboard and the domains of the variables $\{1, \ldots, mn\}$ denote the order of a move sequence. There are constraints to ensure that the knight takes a valid move (i.e., moves either one square horizontally and two squares vertically, or two squares horizontally and one square vertically). Figure 11 gives a solution of the (3,4) instance of the knight's tour problem. The number in each square on the chessboard refers to the sequence of moves. Since the knight's tour problem has many solutions, we soften the problem by assigning a random cost to each allowed binary tuple for soft instances. The random cost is assigned from $\perp$ to mn inclusively. In this problem, the induced model $\mathcal{P}'$ is a better model than $\mathcal{P}$. Therefore, we use $\mathcal{P}'$ as a basic model instead of $\mathcal{P}$.

n
m

10	7	2	5
1	4	11	8
12	9	6	3

Figure 11: A solution of the (3,4) instance of the knight's tour problem

Tables 2, 3, 4, and 5 show the results of classical and soft knight's tour problem using different value ordering heuristics. For both classical and soft instances, enforcing local consistencies on $\mathcal{P}^c$ (columns 5–8) achieves smaller number of fails and shorter runtime than those on $\mathcal{P}$ (colums 2–4). For the instance $(3,14)$, using AC*, FDAC*, EDAC*, and even 2-AC$_c^*$ cannot solve the problem before timeout, but using LB(2,AC*), LB(2,FDAC*), and LB(2,EDAC*) can. In all four tables, LB(2,FDAC*) always achieves the fastest runtime among the local consistencies on $\mathcal{P}^c$. In classical cases (Tables 2 and 4), LB(2,AC*),

Table 3: Experimental results on solving soft knight's tour problem using *smallest-cost-first* value ordering heuristic on $\mathcal{P}$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
(m,n)	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
(3,5)	685	**0.08**	672	0.11	683	0.19	**210**	0.09	**210**	0.09	**210**	0.09	**210**	0.12
(3,6)	4482	0.52	4418	0.81	4463	1.46	**816**	0.39	**816**	**0.37**	**816**	0.43	**816**	0.62
(3,7)	21005	2.94	13629	3.85	13785	5.74	4530	2.11	4716	**2.09**	3551	2.41	**3512**	3.47
(3,8)	112979	19.93	61182	24.30	60967	34.70	21120	11.41	22715	**11.40**	14118	12.45	**13821**	17.78
(3,9)	679347	140.38	341322	166.53	339664	238.65	112080	**72.27**	121708	72.39	73618	79.93	**72429**	118.45
(3,10)	3822955	952.94	1720858	1061.83	1697349	1505.17	507308	433.16	554289	**424.66**	298148	436.09	**293289**	651.64
(3,11)	-	-	-	-	-	-	2310423	2587.76	2586888	2580.80	1234370	**2458.09**	**1214243**	3607.97
(4,5)	45516	4.86	27468	5.85	27374	8.72	5522	1.97	5701	**1.94**	4174	2.14	**4116**	3.06
(4,6)	863270	116.09	415083	129.63	407198	187.90	97222	**42.15**	103934	42.59	61639	43.26	**60354**	63.13
(4,7)	12899196	2335.48	5005880	2339.55	4831495	3248.17	1339779	811.10	1464866	813.16	719474	**745.64**	**702176**	1089.04

Table 4: Experimental results on solving classical knight's tour problem using *dual-smallest-domain-first* value ordering heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
(m,n)	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
(3,5)	672	0.07	672	0.07	672	0.10	**210**	0.08	**210**	0.08	**210**	**0.06**	**210**	0.08
(3,6)	4416	0.45	4416	0.53	4416	0.70	**816**	0.36	**816**	0.34	**816**	**0.29**	**816**	0.33
(3,7)*	545	**0.06**	545	0.07	545	0.10	**158**	0.08	**158**	0.08	**158**	0.08	**158**	0.08
(3,8)*	2657	0.33	2657	0.38	2657	0.50	**732**	0.35	**732**	0.33	**732**	**0.29**	**732**	0.33
(3,9)*	11535	1.66	11535	1.94	11535	2.61	**2005**	1.29	**2005**	1.21	**2005**	**1.07**	**2005**	1.20
(3,10)*	72183	11.79	72183	13.74	72183	18.72	**10628**	8.25	**10628**	7.71	**10628**	**6.72**	**10628**	7.49
(3,11)*	13225	2.17	13225	2.46	13225	3.39	**1878**	1.79	**1878**	1.60	**1878**	**1.31**	**1878**	1.48
(3,12)*	2349445	467.80	2349445	550.14	2349445	743.09	**212657**	269.15	**212657**	240.85	**212657**	**201.84**	**212657**	228.67
(3,13)*	766731	172.10	766731	202.52	766731	274.80	**49684**	94.08	**49684**	83.70	**49684**	**68.33**	**49684**	74.73
(3,14)*	-	-	-	-	-	-	-	-	**3679309**	6816.84	**3679309**	**5550.03**	**3679309**	6249.42
(3,15)*	1498214	367.45	1498214	426.99	1498214	588.97	**58889**	147.85	**58889**	134.74	**58889**	**107.39**	**58889**	118.52
(4,5)*	336	**0.03**	336	**0.03**	336	0.04	**77**	0.04	**77**	**0.03**	**77**	0.04	**77**	0.04
(4,6)*	7914	0.68	7914	0.76	7914	1.04	**1415**	0.56	**1415**	0.53	**1415**	**0.46**	**1415**	0.51
(4,7)*	40344	4.19	40344	4.62	40344	6.55	**5526**	2.70	**5526**	2.51	**5526**	**2.07**	**5526**	2.30
(4,8)*	1183823	145.18	1183823	161.48	1183823	228.35	**76661**	53.58	**76661**	49.00	**76661**	**39.74**	**76661**	43.63
(4,9)*	462506	67.28	462506	74.85	462506	105.82	**33575**	31.86	**33575**	28.07	**33575**	**23.35**	**33575**	25.66
(4,10)*	29317520	5166.23	29317520	5769.60	-	-	**971162**	1469.12	**971162**	1316.55	**971162**	**1034.83**	**971162**	1144.33

Table 5: Experimental results on solving soft knight's tour problem *dual-smallest-domain-first* value ordering heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
(m,n)	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
(3,5)	685	**0.08**	672	0.11	683	0.19	**210**	0.10	**210**	0.09	**210**	0.09	**210**	0.12
(3,6)	4482	0.52	4418	0.81	4463	1.46	**816**	0.39	**816**	**0.37**	**816**	0.43	**816**	0.62
(3,7)	21005	2.94	13629	3.85	13785	5.74	4576	2.13	4750	**2.09**	3571	2.42	**3536**	3.49
(3,8)	112979	19.93	61182	24.30	60967	34.70	21452	**11.46**	22971	11.48	14320	12.57	**14022**	17.96
(3,9)	679347	140.38	341322	166.53	339664	238.65	113032	72.89	121562	**72.20**	73220	79.67	**71996**	118.24
(3,10)	3822955	952.94	1720858	1061.83	1697349	1505.17	511042	431.53	557168	**423.39**	299257	440.19	**294434**	651.79
(3,11)	-	-	-	-	-	-	2319839	2587.78	2590939	2579.96	1236203	**2432.43**	**1216082**	3579.21
(4,5)	45516	4.86	27468	5.85	27374	8.72	5515	1.96	5719	**1.94**	4165	2.14	**4111**	3.05
(4,6)	863270	116.09	415083	129.63	407198	187.90	98151	**42.53**	104256	42.71	61805	43.33	**60522**	63.25
(4,7)	12899196	2335.48	5005880	2339.55	4831495	3248.17	1344421	815.54	1468226	814.30	719927	**742.79**	**702569**	1085.02

LB(2,FDAC*), LB(2,EDAC*), and 2-AC$_c^*$ on $\mathcal{P}^c$ have the same number of fails because EDAC* and FDAC* both degenerate to AC* in classical cases. In soft cases (Tables 3 and 5), LB(2,AC*) and 2-AC$_c^*$ have similar runtime but 2-AC$_c^*$ has a slightly smaller number of fails because of its stronger propagation strength by Theorem 5.3. LB(2,EDAC*) achieves the smallest number of fails in most instances, while LB(2,FDAC*) has the shortest runtime in large and difficult instances.

6.2 Langford's Problem

Recall the Langford's problem (prob024 in CSPLib [19]), denoted as (m, n)-Langford's problem, which is to find an $m \times n$ digit sequence consisting of digits 1 to n, each occurring m times, such that any two consecutive occurrences

Table 6: Experimental results on solving classical Langford's problem using *smallest-cost-first* value ordering heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
(m,n)	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
(2,5)	22	0.01	22	0.01	22	**0.00**	**13**	**0.00**	**13**	0.01	**13**	0.01	**13**	0.01
(2,6)	80	**0.01**	80	**0.01**	80	**0.01**	**57**	**0.01**	**57**	0.02	**57**	**0.01**	**57**	0.02
(2,7)*	**1**	0.01	**1**	0.01	**1**	**0.00**	2	0.01	2	**0.00**	2	**0.00**	2	0.01
(2,8)*	24	0.01	24	**0.00**	24	0.01	**0**	0.01	**0**	0.01	**0**	0.01	**0**	0.01
(2,9)	8576	0.85	8576	**0.81**	8576	1.00	**4147**	0.91	**4147**	0.93	**4147**	1.06	**4147**	1.20
(2,10)	48048	5.05	48048	**4.81**	48048	5.98	**21910**	5.16	**21910**	5.30	**21910**	6.00	**21910**	6.79
(2,11)*	11	**0.01**	11	**0.01**	11	**0.01**	**3**	0.02	**3**	**0.01**	**3**	0.02	**3**	0.03
(2,12)*	6	0.02	6	**0.01**	6	**0.01**	**0**	0.02	**0**	0.02	**0**	0.03	**0**	0.03
(2,13)	14730844	1768.02	14730844	1689.34	14730844	2153.56	**5605943**	**1536.87**	**5605943**	1584.62	**5605943**	1785.55	**5605943**	2032.75
(3,5)	6	**0.00**	6	0.01	6	**0.00**	**3**	**0.00**	**3**	**0.00**	**3**	0.01	**3**	**0.00**
(3,6)	20	**0.01**	20	**0.01**	20	**0.01**	**12**	0.02	**12**	0.02	**12**	0.02	**12**	0.02
(3,7)	62	0.05	62	**0.03**	62	0.04	**29**	0.05	**29**	0.04	**29**	0.04	**29**	0.05
(3,8)	238	0.13	238	**0.10**	238	0.13	**89**	0.15	**89**	0.13	**89**	0.14	**89**	0.17
(3,9)*	192	0.12	192	**0.09**	192	0.12	**41**	0.10	**41**	**0.09**	**41**	0.11	**41**	0.13
(3,10)*	569	0.40	569	0.30	569	0.39	**114**	0.27	**114**	**0.25**	**114**	0.30	**114**	0.34
(3,11)	14512	10.08	14512	7.59	14512	10.00	**2866**	7.15	**2866**	**5.98**	**2866**	6.88	**2866**	7.78
(3,12)	62016	45.72	62016	34.87	62016	47.56	**11729**	29.91	**11729**	**26.89**	**11729**	30.50	**11729**	34.30
(3,13)	300800	247.89	300800	190.08	300800	256.89	**43268**	133.15	**43268**	**115.67**	**43268**	134.06	**43268**	149.87
(3,14)	1368322	1203.69	1368322	926.63	1368322	1231.83	**182304**	628.63	**182304**	**542.30**	**182304**	621.15	**182304**	706.85
(3,15)	7515260	6932.18	7515260	5318.65	-	-	**814604**	3311.54	**814604**	**2770.87**	**814604**	3208.12	**814604**	3604.11

Table 7: Experimental results on solving soft Langford's problem using *smallest-cost-first* value ordering heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
(m,n)	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
(2,5)	75	**0.00**	73	0.01	72	0.00	56	0.01	57	0.01	53	0.01	**53**	0.01
(2,6)	181	**0.01**	178	0.01	177	0.01	130	0.02	132	0.02	125	0.02	**125**	0.02
(2,7)*	95	**0.01**	**84**	0.01	85	0.01	89	0.01	99	0.02	88	0.02	86	0.02
(2,8)*	164	**0.01**	149	0.02	151	0.02	137	0.03	146	0.03	134	0.03	**133**	0.04
(2,9)	10850	1.09	10822	**1.06**	10823	1.06	4970	1.14	4979	1.17	4962	1.32	**4962**	1.34
(2,10)	59683	6.43	59659	6.17	59660	**6.17**	**25165**	6.26	25205	6.42	25195	7.26	25191	7.27
(2,11)*	608	0.07	557	**0.07**	557	0.09	384	0.11	412	0.11	381	0.13	**374**	0.14
(2,12)*	783	0.10	702	**0.10**	700	0.12	**557**	0.18	611	0.19	573	0.22	566	0.25
(2,13)	18155977	2226.49	18155834	2139.47	18155832	2158.92	**6325529**	**1798.40**	6325636	1850.45	6325559	2085.56	6325540	2084.30
(3,5)	368	**0.04**	286	**0.04**	279	0.05	265	0.07	327	0.08	258	0.09	**257**	0.11
(3,6)	900	0.14	683	**0.12**	670	0.15	563	0.20	672	0.22	539	0.26	**526**	0.30
(3,7)	2242	0.51	1658	**0.42**	1647	0.52	1455	0.73	1743	0.81	1384	0.91	**1355**	1.05
(3,8)	2744	0.82	1985	**0.62**	**1975**	0.76	2275	1.33	3013	1.59	2141	1.68	2107	1.93
(3,9)*	2603	0.52	**1163**	**0.31**	1360	0.44	3680	2.13	6173	3.37	4259	3.66	4180	4.11
(3,10)*	6834	1.79	4074	**1.35**	**3964**	1.62	8279	5.85	10317	7.15	7319	7.75	7183	8.59
(3,11)	76579	44.00	69117	**34.23**	68996	42.54	**27916**	39.69	33633	44.11	30053	50.58	29671	54.67
(3,12)	274564	180.14	253984	**138.45**	253471	173.40	80282	141.02	89105	140.19	75818	159.09	**75354**	171.50
(3,13)	946260	734.88	920950	576.89	920173	728.34	**173636**	**420.16**	220283	451.17	190016	508.92	189261	547.18
(3,14)	4388875	3703.48	4317762	2891.87	4317129	3639.71	**568250**	**1724.69**	690449	1768.77	648979	2060.19	646147	2238.74

of digit i's are separated by i other digits. In the classical Langford's problem, many instances are over-constrained. For example, only the $(2,7)$, $(2,8)$, $(2,11)$, $(2,12)$, $(3,9)$, and $(3,10)$ instances are satisfiable among $(2,n)$, where $5 \leq n \leq 13$, and $(3,n)$, where $5 \leq n \leq 15$. Therefore, we soften the problem, as described in Section 4.1, by allowing violation of constraints (except the all-different constraint) at random costs uniformly from 1 to $\top$ inclusively. After softening the constraints, there can be more solutions for a model and an unsatisfiable model can become satisfiable.

Tables 6, 7, 8, and 9 show the results on various (m,n) instances of classical and soft Langford's problem using different value ordering heuristics. Among all local consistencies, solving a combined model $\mathcal{P}^c$ achieves smaller number of fails in most instances, not to mention the preliminary results in Table 1. This shows that 2-AC$_c^*$, LB(2,AC*), LB(2,FDAC*), and LB(2,EDAC*) do far more prunings than AC*, FDAC*, and EDAC*. Thus, more search space can be reduced, especially for the larger and more difficult instances, which require larger amount of search efforts to either prove unsatisfiability or find an optimal solution. We improve the number of fails and runtime of, say, $(3,14)$ instance

Table 8: Experimental results on solving classical Langford's problem using *dual-smallest-domain-first* value ordering heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
(m,n)	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
(2,5)	22	0.01	22	0.01	22	**0.00**	**13**	**0.00**	**13**	0.01	**13**	0.01	**13**	0.01
(2,6)	80	**0.01**	80	**0.01**	80	**0.01**	**57**	**0.01**	**57**	**0.01**	**57**	**0.01**	**57**	0.02
(2,7)*	**1**	0.01	**1**	0.01	**1**	**0.00**	4	0.01	4	0.01	4	0.01	4	0.01
(2,8)*	24	0.01	24	**0.00**	24	0.01	**6**	0.01	**6**	0.01	**6**	0.01	**6**	0.01
(2,9)	8576	0.85	8576	**0.81**	8576	1.00	**4147**	0.90	**4147**	0.92	**4147**	1.06	**4147**	1.20
(2,10)	48048	5.05	48048	**4.81**	48048	5.98	**21910**	5.16	**21910**	5.26	**21910**	5.95	**21910**	6.76
(2,11)*	11	**0.01**	11	**0.01**	11	**0.01**	**1**	0.02	**1**	**0.01**	**1**	0.03	**1**	0.02
(2,12)*	**6**	0.02	**6**	**0.01**	**6**	**0.01**	15	0.03	15	0.03	15	0.04	15	0.04
(2,13)	14730844	1768.02	14730844	1689.34	14730844	2153.56	**5605943**	**1527.87**	**5605943**	1578.14	**5605943**	1779.36	**5605943**	2024.38
(3,5)	6	**0.00**	6	0.01	6	**0.00**	**3**	**0.00**	**3**	0.01	**3**	0.01	**3**	0.01
(3,6)	20	**0.01**	20	**0.01**	20	**0.01**	**12**	0.02	**12**	0.02	**12**	0.02	**12**	0.02
(3,7)	62	0.05	62	**0.03**	62	0.04	**29**	0.05	**29**	0.04	**29**	0.05	**29**	0.05
(3,8)	238	0.13	238	**0.10**	238	0.13	**89**	0.14	**89**	0.13	**89**	0.15	**89**	0.16
(3,9)*	192	0.12	192	**0.09**	192	0.12	**47**	0.12	**47**	0.11	**47**	0.12	**47**	0.13
(3,10)*	569	0.40	569	0.30	569	0.39	**90**	0.25	**90**	**0.22**	**90**	0.25	**90**	0.29
(3,11)	14512	10.08	14512	7.59	14512	10.00	**2866**	7.14	**2866**	**6.24**	**2866**	7.09	**2866**	7.85
(3,12)	62016	45.72	62016	34.87	62016	47.56	**11729**	29.85	**11729**	**27.33**	**11729**	30.84	**11729**	34.92
(3,13)	300800	247.89	300800	190.08	300800	256.89	**43268**	141.13	**43268**	**116.60**	**43268**	133.51	**43268**	150.52
(3,14)	1368322	1203.69	1368322	926.63	1368322	1231.83	**182304**	641.70	**182304**	**548.96**	**182304**	621.25	**182304**	709.33
(3,15)	7515260	6932.18	7515260	5318.65	-	-	**814604**	3452.07	**814604**	**2772.46**	**814604**	3186.62	**814604**	3668.44

Table 9: Experimental results on solving soft Langford's problem using *dual-smallest-domain-first* value ordering heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
(m,n)	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
(2,5)	75	**0.00**	73	0.01	72	0.00	63	0.01	64	0.01	**58**	0.01	59	0.01
(2,6)	181	**0.01**	178	0.01	177	0.01	151	0.02	151	0.02	145	0.03	**144**	0.03
(2,7)*	95	**0.01**	**84**	0.01	85	0.01	186	0.03	194	0.03	173	0.04	172	0.04
(2,8)*	164	**0.01**	**149**	0.02	151	0.02	428	0.08	435	0.09	407	0.09	404	0.10
(2,9)	10850	1.09	10822	**1.06**	10823	1.06	5234	1.19	5248	1.22	5179	1.37	**5176**	1.41
(2,10)	59683	6.43	59659	6.17	59660	**6.17**	26045	6.46	26063	6.62	25946	7.48	**25943**	7.50
(2,11)*	608	0.07	557	**0.07**	**557**	0.09	20148	5.20	20175	5.28	19955	5.93	19978	5.97
(2,12)*	783	0.10	702	**0.10**	**700**	0.12	107242	29.17	107281	30.01	108521	34.56	108389	34.93
(2,13)	18155977	2226.49	18155834	2139.47	18155832	2158.92	6375033	**1832.26**	6375074	1864.47	**6371007**	2110.17	6372172	2099.90
(3,5)	368	**0.04**	286	**0.04**	**279**	0.05	416	0.10	444	0.10	336	0.11	330	0.14
(3,6)	900	0.14	683	**0.12**	**670**	0.15	1011	0.34	1069	0.34	800	0.38	786	0.44
(3,7)	2242	0.51	1658	**0.42**	**1647**	0.52	3052	1.42	3244	1.37	2301	1.48	2262	1.68
(3,8)	2744	0.82	1985	**0.62**	**1975**	0.76	5917	3.32	6298	3.21	4356	3.39	4298	3.84
(3,9)*	2603	0.52	**1163**	**0.31**	1360	0.44	14251	9.36	15211	9.01	10445	9.36	10316	10.53
(3,10)*	6834	1.79	4074	**1.35**	**3964**	1.62	29169	24.20	30773	22.86	24219	26.16	23868	29.32
(3,11)	76579	44.00	69117	**34.23**	68996	42.54	80521	101.38	83418	93.25	67176	103.07	**66492**	113.18
(3,12)	274564	180.14	253984	**138.45**	253471	173.40	218142	343.92	225727	312.39	178717	336.07	**176423**	360.53
(3,13)	946260	734.88	920950	**576.89**	920173	728.34	439224	901.46	451479	799.07	367232	856.07	**364754**	948.65
(3,14)	4388875	3703.48	4317762	**2891.87**	4317129	3639.71	1346922	3535.94	1365833	3039.68	1201030	3377.41	**1194703**	3658.62

in Table 6, by factors of 7.5 and 1.9 respectively on average. There are even instances, the $(3,15)$ instances in Tables 6 and 8, which cannot be solved by enforcing EDAC* on a single model within the time limit, but enforcing 2-AC$_c^*$, LB(2,AC*), LB(2,FDAC*), and LB(2,EDAC*) on the combined model can solve. The reduction ratios of number of fails and runtime of combined models to single models increase with the problem size on average. Exceptions are those marked with "*," which can terminate when a zero cost solution is found. Such instances require relatively fewer search efforts, and the overhead of an extra model may not be compensated in these cases. In addition, we observe that using LB(2,Φ) is not always faster than using 2-AC$_c^*$, although both are much better than using single models. In $(3,n)$ classical and soft instances, LB(2,AC*) and LB(2,FDAC*) have faster runtime than 2-AC$_c^*$, but they are slightly slower than 2-AC$_c^*$ in the $(2,n)$ classical and soft instances.

Table 10: Experimental results on solving classical Latin square problem using *smallest-cost-first* value ordering heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
n	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
5*	**0**	**0.00**	**0**	0.01	**0**	0.01	**0**	0.01	**0**	0.01	**0**	0.01	**0**	0.01
10*	1	**0.12**	1	**0.12**	1	0.15	**0**	0.31	**0**	0.29	**0**	0.31	**0**	0.35
15*	14	**1.56**	14	1.59	14	2.04	**0**	3.72	**0**	3.16	**0**	3.27	**0**	4.05
20*	645	**9.48**	645	13.75	645	17.23	**0**	21.58	**0**	17.55	**0**	17.38	**0**	22.18

Table 11: Experimental results on solving soft Latin square problem using *smallest-cost-first* value ordering heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
n	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
5	25023	2.64	10039	2.87	7319	5.08	8397	**2.21**	21866	5.60	8942	6.87	**6757**	12.26
6	751412	154.56	111545	106.44	76353	144.16	93109	**58.22**	646190	304.12	104810	256.24	**71865**	359.24
7	-	-	1866490	3602.15	**950480**	3648.19	1392309	**1384.49**	-	-	-	-	-	-

6.3 Latin Square Problem

The Latin square problem (prob003 in CSPLib [19]) is to fill an $n \times n$ matrix with n numbers such that each column and each row must form a permutation. We use a standard model [13] as the single model $\mathcal{P}_1$ which uses the set of variables $\mathcal{X} = \{x_{ij} \mid 1 \leq i, j \leq n\}$ for the cells of the matrix. Each variable x_{ij} represents the cell in the i-th row and j-th column and has a domain $D_{x_{ij}} = \{1, \ldots, n\}$ denoting the possible numbers to be filled in a cell. There are $O(n^3)$ disequality constraints $x_{ij} \neq x_{il}$ for $1 \leq i \leq n$ and $1 \leq j < l \leq n$ to ensure that no two cells in the same row take the same number. Similarly, there are $O(n^3)$ disequality constraints $x_{ij} \neq x_{lj}$ for $1 \leq j \leq n$ and $1 \leq i < l \leq n$ to ensure that no two cells in the same column take the same number. We initialize each binary cost which violates a disequality constraint to $\top$ and all other costs to $\perp$. Although $\mathcal{P}_1$ is not a permutation WCSP, it is a vector of n permutation WCSPs such that each row and column is a permutation.

Similar to the Langford's problem, there are different ways to model the Latin square problem [13]. We can use the set of variables $\mathcal{Y} = \{y_{ik} \mid 1 \leq i, k \leq n\}$, where each variable y_{ik} represents the number k in the i-th row. The domain of each variable is $D_{y_{ik}} = \{1, \ldots, n\}$ denoting the possible column positions for the number k in the i-th row. Based on this viewpoint $(\mathcal{Y}, \mathcal{D}_\mathcal{Y})$ and the bijective function $f(x_{ij} \mapsto k) = y_{ik} \mapsto j$ for all $i, j, k \in \{1, \ldots, n\}$, the induced model can be generated automatically by generalized model induction. Contrary to the Langford's problem, the Latin square problem has many solutions for each instance. In many situations, we may want to have preferences among solutions. Therefore, to model such situation, we change each binary $\perp$ cost in the constraints of the model to a random cost from $\perp$ to n inclusively. In this soft Latin square problem, the problem is tightened and the number of solutions is smaller.

Tables 10, 11, 12, and 13 show the experimental results of the classical and soft Latin square problems using different value ordering heuristics. We can see from Tables 10 and 12 that classical Latin square problems are easy to solve up to $n = 20$. The amount of search is small even using a single model.

Table 12: Experimental results on solving classical Latin square problem using *dual-smallest-domain-first* value ordering heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
n	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
5*	**0**	**0.00**	**0**	0.01	**0**	0.01	**0**	0.01	**0**	**0.00**	**0**	**0.00**	**0**	0.01
10*	1	**0.12**	1	**0.12**	1	0.15	**0**	0.33	**0**	0.29	**0**	0.29	**0**	0.36
15*	14	**1.56**	14	1.59	14	2.04	**0**	3.95	**0**	3.11	**0**	3.20	**0**	4.48
20*	645	**9.48**	645	13.75	645	17.23	**0**	22.28	**0**	18.18	**0**	18.32	**0**	25.07

Table 13: Experimental results on solving soft Latin square problem using *dual-smallest-domain-first* value ordering heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
n	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
5	25023	2.64	10039	2.87	7319	5.08	8353	**2.19**	21853	5.57	8879	6.83	**6679**	12.15
6	751412	154.56	111545	106.44	76353	144.16	93109	**58.23**	646190	303.36	104810	256.21	**71865**	359.09
7	-	-	1866490	3602.15	**950480**	3648.19	1392309	**1378.96**	-	-	-	-	-	-

Nonetheless, enforcing 2-AC$_c^*$, LB(2,AC*), LB(2,FDAC*), and LB(2,EDAC*) on $\mathcal{P}^c$ can still reduce the search space to achieve no backtracks, although their runtime are larger due to the overhead of an extra model. However, the soft Latin square problems, as shown in Tables 11 and 13, are much more difficult than the classical ones, since we are searching for the most preferred solution Therefore, we can solve only the smaller instances within the time limit of two hours.

Tables 11 and 13 show that 2-AC$_c^*$ achieves the fastest runtime, while AC*, LB(2,AC*), LB(2,FDAC*), and LB(2,EDAC*) cannot solve the problem within two hours. On the other hand, FDAC* and EDAC* have higher time complexities $O(n^3d^3)$ and $O(n^2d^2 \max\{nd, \top\})$ respectively, which can possibly remove more domain values and tighten the global lower bound. Thus, the time gained from fewer number of backtracks cannot compensate that used in performing EDAC* and FDAC* algorithms. The same reason explains the results of enforcing LB(2,FDAC*) and LB(2,EDAC*). The 2-AC$_c^*$ is striking a good balance in soft Latin square problems between the amount of pruning and the time spent on discovering the values for pruning.

6.4 n-Queens Problem

The n-queens problem is to place n queens on an $n \times n$ chessboard so that no two queens are placed on the same row, same column, or same diagonal. To model the problem into a WCSP $\mathcal{P}$, we use n variables $\mathcal{X} = \{x_1, \ldots, x_n\}$. Each variable x_i denotes the row position of queen i in column i of the chessboard. The domains of the variables are thus $\{1, \ldots, n\}$. Since the n-queens problem has many solutions, we assert preferences among the solutions by assigning to each allowed binary tuple a random cost from $\perp$ to $\top$ inclusive for soft instances.

Tables 14, 15, 16, and 17 shows the experimental results of classical and soft n-queens problems using different value ordering heuristics. In classical cases, the cost can be either 0 or 1 only. The smallest-cost-first value ordering heuristic is thus the same as the smallest-value-first ordering heuristic, which does not favour the classical n-queens problem, especially for the combined models. Table

Table 14: Experimental results on solving classical n-queens problem using *smallest-cost-first* value ordering heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
n	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
80*	**9**	2.05	**9**	**1.75**	**9**	2.23	10321	12.15	10321	12.18	10321	11.57	10321	13.34
85*	**6**	**2.21**	**6**	2.26	**6**	2.37	-	-	-	-	-	-	-	-
90*	**199**	**2.76**	**199**	3.45	**199**	3.59	-	-	-	-	-	-	-	-
95*	**1345**	**3.60**	**1345**	3.68	**1345**	4.70	-	-	-	-	-	-	-	-
100*	**13**	**4.28**	**13**	4.35	**13**	4.55	-	-	-	-	-	-	-	-
105*	**30564**	**11.41**	**30564**	12.20	**30564**	14.68	-	-	-	-	-	-	-	-
110*	**6693868**	**1245.09**	**6693868**	1433.44	**6693868**	1932.29	-	-	-	-	-	-	-	-
115*	-	-	-	-	-	-	-	-	-	-	-	-	-	-
120*	**3797818**	**994.78**	**3797818**	1080.30	**3797818**	1488.92	-	-	-	-	-	-	-	-

Table 15: Experimental results on solving soft n-queens problem using *smallest-cost-first* value order heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
n	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
15*	5608	**0.46**	5044	0.50	4959	0.63	2695	0.57	2740	0.57	2485	0.64	**2432**	0.75
16*	10611	**0.96**	9566	1.03	9382	1.29	5426	1.24	5516	1.23	4939	1.39	**4822**	1.63
17*	17369	**1.65**	15139	1.80	14703	2.25	6658	1.68	6778	1.66	6038	1.89	**5880**	2.23
18*	58375	**6.23**	51073	6.76	49309	8.29	24126	7.07	24495	6.88	22370	7.78	**21796**	9.10
19*	81022	**9.20**	70179	10.05	67341	12.41	32533	9.65	33324	9.43	28783	10.90	**27469**	12.61
20*	172220	**21.65**	150939	23.50	145062	28.94	87419	29.31	89193	28.53	79209	32.98	**76146**	38.55
21*	535145	73.28	463225	79.78	441403	97.35	178652	63.75	183176	**61.71**	157600	71.57	**149083**	82.30
22*	1287717	196.07	1130132	211.09	1078297	257.76	459420	189.23	468781	**181.04**	418087	206.83	**400026**	239.30
23*	4780028	810.60	4256142	868.88	4076254	1060.10	1286071	610.17	1307948	**577.06**	1188221	653.88	**1143529**	757.32
24*	11079154	2042.66	9928478	2182.87	9518203	2637.39	3222029	1675.23	3276754	**1571.79**	2978541	1777.95	**2861661**	2041.45

Table 16: Experimental results on solving classical n-queens problem using *dual-smallest-domain-first* value order heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
n	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
80*	9	2.05	9	**1.75**	9	2.23	**0**	4.37	**0**	3.92	**0**	3.72	**0**	4.10
85*	6	**2.21**	6	2.26	6	2.37	**0**	6.46	**0**	4.40	**0**	4.82	**0**	5.33
90*	199	**2.76**	199	3.45	199	3.59	**0**	7.41	**0**	6.19	**0**	6.27	**0**	7.00
95*	**1345**	**3.60**	**1345**	3.68	**1345**	4.70	-	-	-	-	-	-	-	-
100*	13	**4.28**	13	4.35	13	4.55	**1**	11.78	**1**	10.97	**1**	11.08	**1**	9.53
105*	30564	11.41	30564	12.20	30564	14.68	**0**	13.58	**0**	13.79	**0**	**11.33**	**0**	14.45
110*	6693868	1245.09	6693868	1433.44	6693868	1932.29	**0**	**16.80**	**0**	16.86	**0**	16.83	**0**	17.63
115*	-	-	-	-	-	-	**0**	20.84	**0**	**16.85**	**0**	16.99	**0**	17.67
120*	3797818	994.78	3797818	1080.30	3797818	1488.92	**10**	25.15	**10**	**20.45**	**10**	25.23	**10**	26.31

Table 17: Experimental results on solving soft n-queens problem using *dual-smallest-domain-first* value order heuristic on $\mathcal{P}^c$

	AC*		FDAC*		EDAC*		2-AC$_c^*$		LB(2,AC*)		LB(2,FDAC*)		LB(2,EDAC*)	
n	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
15*	5608	**0.46**	5044	0.50	4959	0.63	2759	0.58	2803	0.58	2546	0.65	**2488**	0.77
16*	10611	**0.96**	9566	1.03	9382	1.29	5340	1.23	5427	1.21	4873	1.37	**4769**	1.61
17*	17369	**1.65**	15139	1.80	14703	2.25	7291	1.85	7429	1.81	6668	2.08	**6493**	2.46
18*	58375	**6.23**	51073	6.76	49309	8.29	26370	7.73	26762	7.48	24479	8.49	**23880**	9.95
19*	81022	**9.20**	70179	10.05	67341	12.41	32921	9.91	33681	9.60	29197	11.07	**27919**	12.81
20*	172220	**21.65**	150939	23.50	145062	28.94	94256	32.44	95941	31.16	86345	35.83	**83442**	42.09
21*	535145	73.28	463225	79.78	441403	97.35	178540	65.39	182731	**62.39**	159087	72.03	**151279**	83.29
22*	1287717	196.07	1130132	211.09	1078297	257.76	470117	200.77	478708	**188.64**	432133	214.43	**415874**	249.10
23*	4780028	810.60	4256142	868.88	4076254	1060.10	1274332	617.77	1294032	**575.07**	1183741	650.64	**1143417**	755.17
24*	11079154	2042.66	9928478	2182.87	9518203	2637.39	3256536	1738.86	3307770	**1602.52**	3029570	1806.44	**2921698**	2081.07

14 reproduces the results shown by Cheng *et al.* [5] in which the dual-smallest-domain-first is a better value ordering heuristic for n-queens problem and gives much better results than the smallest-cost-first (smallest-value-first) ordering heuristic. In soft cases, solving a combined model $\mathcal{P}^c$ achieves fewer number of fails and mostly faster runtime than solving a single model $\mathcal{P}$, which means that 2-AC$_c^*$, LB(2,AC*), LB(2,FDAC*), and LB(2,EDAC*) do more prunings

than AC*, FDAC*, and EDAC* and reduce more search space. LB(2,AC*) is the most efficient for the larger instances. Although 2-AC_c^* is stronger than LB(2,AC*), it gives a longer runtime than LB(2,AC*) even it achieves smaller number of fails in solving the soft instances. This is mainly due to the overhead of cost projection transmitting between sub-models in 2-AC_c^*. LB(2,EDAC*) always has the fewest number of fails since EDAC* is stronger than AC* and FDAC*. However, the runtime of LB(2,EDAC*) is not the fastest, as the extra prunings cannot compensate for the overhead of EDAC*.

6.5 Discussions

In the experiments, we have used two different value ordering heuristics: the smallest-cost-first and the dual-smallest-domain-first. The dual-smallest-domain-first heuristic is only applicable for redundant modeling and is shown to be especially suitable for the n-queens problem [5]. From the results, we observe that using different value ordering heuristics results differently. The smallest-cost-first heuristic favors our first approach, while the dual-smallest-domain-first heuristic favors our second approach. This also means that the results are sensitive to the value ordering heuristics.

Our results show that enforcing 2-AC_c^*, LB(2,AC*), LB(2,FDAC*), and LB(2,EDAC*) on combined models outperform enforcing AC*, FDAC*, and EDAC* on single models for the larger and more difficult instances, which require large amount of search efforts to either prove unsatisfiability or find an optimal solution. Besides, enforcing 2-AC_c^*, LB(2,AC*), LB(2,FDAC*), and LB(2,EDAC*) on combined models result in a shorter runtime to solve those problems, with the exception of the *classical* Latin square problem which requires relatively small amount of search efforts as there are many solutions in the model. Though enforcing local consistencies on combined models can help to reduce more search space, this cannot counteract the overhead of an extra model, resulting in a larger runtime than those of single models.

Enforcing local consistencies on combined models achieves the smallest number of fails than enforcing consistencies on single models in all cases except in the soft Latin square problem. It even has no backtracks in the classical Latin square problem. This shows that redundant modeling helps to increase constraint propagation by sharing the pruning and cost projection information between sub-models of the combined models. In fact, although we have shown that m-AC_c^* is strictly stronger than AC* and LB(m,AC*), the relative propagation strength among FDAC*, EDAC*, m-AC_c^*, LB(m,Φ) is still not clear. LB(2,Φ) does not always perform better than 2-AC_c^*. However, it has the flexibility of choosing *any* local consistency for single models to solve a problem. The refinements of local consistencies for combined models, other than m-NC_c^* and m-AC_c^*, are still not yet known. More theoretical and empirical studies have to be conducted to have a better understanding on their propagation behavior.

7 Related Work

In this section, we present the research that is related to our work on soft constraints, local consistencies in WCSP, and redundant modeling. We briefly describe various approaches to soft constraint satisfaction problems and give some other existing local consistency notions in WCSP. Next, we present an overview of redundant modeling and channeling constraints.

7.1 Soft Constraint Satisfaction Problems

In classical CSP, all constraints are hard. They can either be satisfied or violated. It is sometimes difficult to use a classical CSP to model real-life problems where there are preferences and costs. Different types of soft CSPs are therefore proposed to solve optimization and over-constrained problems. Some examples are *fuzzy constraint satisfaction problems* (FCSPs) [40], *possibilistic constraint satisfaction problems* (PossCSPs) [42], *probabilistic constraint satisfaction problems* (ProbCSPs) [15], *partial constraint satisfaction problems* (PCSPs) [18] and *weighted constraint satisfaction problems* (WCSPs) [45]. The above soft CSPs can be encapsulated and represented by two meta-frameworks: *valued constraint satisfaction problems* (VCSPs) [44] and *semiring-based constraint satisfaction problems* (SCSPs) [1, 2, 3]. Since our work is focused on WCSP, which is a specific subclass of VCSP, we will present VCSP in more detail.

A *valued constraint satisfaction problem* (VCSP) [44] is a generic soft constraint framework which associates each constraint of a classical CSP with a valuation. The valuations are taken from a totally ordered set of monoid, combined using the monoid operator. They are interpreted as levels of violation by costs, degrees of preference, probabilities, weights, etc. In order to deal with the over-constrained problems, it is necessary to be able to express the fact that a constraint can eventually be violated. This can be done by associating each constraint with a valuation. A *valuation structure* is defined by a tuple $S = (E, \circledast, \succ)$ where E is a set such that its elements are called *valuations*. The valuations are *totally ordered* by $\succ$, with a maximum element denoted by $\top$ and a minimum element denoted by $\bot$. The binary operator $\circledast$ is commutative, associative, and closed under E that satisfies identity, monotonicity, and has an absorbing element in E. A VCSP [44] is a tuple $\mathcal{P} = (\mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X}, \mathcal{S}, \varphi)$, where $\mathcal{X}$ is a set of variables, $\mathcal{D}_\mathcal{X}$ is a set of variable domains, $\mathcal{C}_\mathcal{X}$ is a set of constraints, $\mathcal{S} = (E, \circledast, \succ)$ is a valuation structure, and φ is an *application* from $\mathcal{C}_\mathcal{X}$ to E. The valuation $\varphi(C)$ of $C \in \mathcal{C}_\mathcal{X}$ returns an element in E. A tuple can be evaluated by combining the valuations of all the violated constraints using $\circledast$. Given a VCSP $\mathcal{P} = (\mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X}, \mathcal{S}, \varphi)$ and a tuple θ where $var(\theta) \subset \mathcal{X}$, the valuation of θ is denoted by $\mathcal{V}_\mathcal{P}(\theta)$ such that $\mathcal{V}_\mathcal{P}(\theta) = \circledast\{\varphi(C) \mid C \in \mathcal{C}_\mathcal{X} \wedge var(C) \subset var(\theta) \wedge \theta \text{ violates } C\}$. A *solution* of a valued CSP is to find a complete tuple θ with a minimum valuation according to the order $\succ$.

VCSP is an abstract framework which provides general algorithms and properties [44]. Some other types of CSPs, such as classical CSPs and WCSPs, can be described as an instance of valued CSPs by choosing an appropriate valuation structure. For example, the valuation structure for a classical CSP is given by the boolean lattice $E = \{true, false\}$, where $false = \top \succ true = \bot$, and $\circledast = \wedge$. For WCSPs, the valuation structure corresponds to $E = \mathbb{N} \cup \{+\infty\}$, $0 = \bot$, $+\infty = \top$, and $\circledast = +$, using the $>$ ordering for natural numbers as $\succ$. Schiex [43] extended the notion of local consistency from classical CSPs to VCSPs and showed that the notion provides stronger constraint propagation and gives a better lower bound of the problem.

7.2 Redundant Modeling and Channeling Constraints

Handcrafting multiple classical CSP models for the same problem is common, although not trivial. Nadel [39] took nine models of the n-queens problem to show that there are usually many different CSP models for a problem. Different models of the same problem can have different execution performances. However, it is not trivial to distinguish which model is "better" than another and there is no notion of the "best" model. Cheng *et al.* [6, 7, 8, 5] proposed the concept of redundant modeling. Different models of the same problem can be *combined* using channeling constraints. The combined model can take the advantages of each sub-models to increase constraint propagation and efficiency. Cheng *et al.* used the n-queens problem and a real-life nurse rostering problem to illustrate how to combine two redundant models using channeling constraints to achieve extra constraint propagation and significant speedup. Smith [46, 47] studied redundant modeling on two permutation problems, the n-queens problem and the Langford's problem. She suggested that a dual representation can be obtained by swapping the roles of variables and values. By linking with channeling constraints and removing some constraints, a minimal combined model, which has a much smaller search tree and can speed up problem solving process, can be obtained. Dotú, del Val, and Cebrián [13] also adopted the redundant modeling technique in solving large instances of the Quasigroup Completion Problem in the transition phase region. Hnich, Smith, and Walsh [49, 23, 22] performed an extensive theoretical and empirical study of different models for permutation and injection problems. They showed a general methodology to compare various models by defining a measure of constraint tightness by the level of local consistency being enforced in the models.

Besides redundant modeling, there are some other approaches which can increase the efficiency of constraint solving. One possible approach is to use different techniques concurrently and exchange useful information during search. Marti and Rueher [37] proposed a cooperative architecture which combines the symbolic and numeric solvers based on asynchronous communication between the heterogeneous solvers. Hooker *et al.* [24] proposed a modeling framework in which the constraint programming and linear programming interacted with each other via the conditional constraints. This framework made use of the advantages of either techniques to achieve better results in solving combinatorial optimization problems. Easton, Nemhauser, and Trick [14] implemented an algorithm for the travelling tournament problem. The algorithm was a combination of integer programming and constraint programming, each of which was responsible for different parts of the problem. Montoyo *et al.* [38] combined a knowledge-based method and a corpus-based method to achieve better results in completing the task of word sense disambiguation. Van Hentenryck and Michel [20, 21] studied the nondeterministic control structure for hybrid search procedures and used the job-shop scheduling to illustrate their results. Cotta *et al.* [10] suggested several local search-based hybrid algorithms for solving the Golomb ruler problem. The algorithms are developed using various stochastic methods and systematic techniques. Wallace [48] gave a detailed and comprehensive survey on various techniques in handling constraints with constraint programming. Manisterski, Sarne, and Kraus [36] presented a new search strategy which based on cooperative search with concurrent interactions. They showed how the proposed strategy outperforms the current ones.

8 Concluding Remarks

In this section, we summarize our contributions in this paper and give some possible directions for future research.

8.1 Contributions

We have applied the concept of redundant modeling in permutation WCSPs and proposed two approaches to handle combined WCSP models that contain m sub-models. The contribution of our work can be summarized as follows.

First, redundant modeling requires multiple models of the same problem to be connected by channeling constraints. In classical CSP, though not trivial, it is relatively easy to handcraft an alternative model of a problem. However, it is more difficult to do so for WCSPs since the costs are not only $\perp$ and $\top$. Law and Lee [29, 30] introduced model induction which automatically generates a redundant permutation CSP model using a given one and another viewpoint of the CSP. We have *generalized* the notion to *permutation WCSPs*. Aided with examples, we show how to convert different constraints from one permutation WCSP to another and yield a redundant permutation WCSP automatically.

Second, we have discovered that naively combining redundant WCSP models using channeling constraints and relying on the standard propagation algorithms for the channeling constraints to transmit pruning information between sub-models do not work well. This discovery is supported by our preliminary experiments done on the combined WCSP models to evaluate the performance of single and combined models when enforcing NC* and AC*. By analyzing the propagation behavior in the combined models, we observe the undesirable behavior that enforcing AC* on a combined WCSP model can miss pruning opportunities which are available even in a single model.

Third, through the investigation of the adverse behavior encountered when enforcing NC* and AC* on a combined model, we have generalized NC* and AC*, and proposed m-NC$^*_c$ and m-AC$^*_c$ respectively for combined models containing m sub-models. We have proven that m-NC^*_c and m-AC^*_c are strictly stronger than NC* and AC* respectively. Experiments on our implementations of 2-NC^*_c and 2-AC^*_c have shown the benefits of extra prunings, which lead to a greatly reduced search space and better runtime than the state-of-the-art AC*, FDAC*, and EDAC* algorithms on both classical and soft benchmark problems, especially *hard* instances.

Fourth, while our first approach requires the adaption of some existing local consistencies in order to apply to combined models, in our second approach, we proposed a parameterized local consistency LB(m,Φ), which can be used with *any* number of sub-models and *any* existing or future local consistencies that are targeted for solving single models. This approach makes our proposal highly flexible. Experimental results on several permutation problems confirm that LB(2,Φ) performs well when instantiated with the state-of-the-art AC*, FDAC*, and EDAC*. The search space is significantly reduced when compared with using single models. LB(2,Φ) is also competitive with, if not better than, 2-AC^*_c in the benchmarks.

8.2 Future Work

Our work extends the concept of redundant modeling from classical CSPs to WCSPs. Since redundant modeling in WCSPs is a new concept, there is plenty of scope for future work.

First, the proposed generalized model induction is currently applied to generate a dual model of a permutation WCSP. It is interesting to investigate how general induced WCSP models can be generated. We can check if the same techniques of generalized model induction can be applied to non-permutation WCSPs and what requirements of the channeling constraints for model induction are needed.

Second, suppose we are able to obtain redundant WCSP models for non-permutation WCSPs, it is interesting to study how to combine them. Given a non-permutation WCSP model, its redundant model can have either set variables [33, 34] or Boolean variables (i.e., variables with domain $\{0, 1\}$.) Thus, the combined model is a combination of different type of variables. Lee and Siu [33, 34] proposed a framework and defined various consistency notions for WCSPs with set variables. It is worthwhile to study how integer and set sub-models can be connected to enhance constraint propagation.

Third, in our first approach, we have refined node and arc consistencies for combined models. There are other existing local consistency notions in WCSPs, such as FDAC* [27] and EDAC* [11]. It would be also interesting to incorporate c-supports and c-projections to FDAC* and EDAC* to obtain m-$\mathrm{FDAC}^*_\mathrm{c}$ and m-$\mathrm{EDAC}^*_\mathrm{c}$ respectively.

Acknowledgements

We would like to thank Javier Larrosa, Irwin King, and Evangeline Young, and the anonymous referees from AI 2006, AI 2007, and the *Constraints* journal for their constructive comments. The work described in this paper was substantially supported by grants (CUHK413207 and CUHK413808) from the Research Grants Council of Hong Kong SAR.

References

[1] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs: Basic properties and comparison. In *Over-Constrained Systems*, volume 1106, pages 111–150, 1996.

[2] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

[3] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints*, 4(3):199–240, 1999.

[4] S. Bouveret, F. Heras, S. de Givry, J. Larrosa, M. Sanchez, and T. Schiex. ToolBar: a state-of-the-art platform for WCSP. Technical report, http://www.inra.fr/bia/T/degivry/ToolBar.pdf, 2005.

[5] 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.

[6] B.M.W. Cheng, J.H.M. Lee, and J.C.K. Wu. A constraint-based nurse rostering system using a redundant modeling approach. In *Proceedings of the 8th International Conference on Tools with Artificial Intelligence*, pages 140–148, 1996.

[7] B.M.W. Cheng, J.H.M. Lee, and J.C.K. Wu. Speeding up constraint propagation by redundant modeling. In *Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming*, pages 91–103, 1996.

[8] 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.

[9] B.Y. Choueiry, B. Faltings, and G. Noubir. Abstraction Methods for Resource Allocation. In *Proceedings of the Workshop on Theory Reformulation and Abstraction*, pages 2–71/2–90, 1994.

[10] C. Cotta, I. Dotú, A.J. Fernández, and P. Van Hentenryck. Local search-based hybrid algorithms for finding golomb rulers. *Constraints*, 12(3):263–291, 2007.

[11] S. de Givry, F. Heras, M. Zytnicki, and J. Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 84–89, 2005.

[12] R. Debruyne and C. Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 412–417, 1997.

[13] I. Dotú, A. del Val, and M. Cebrián. Redundant modeling for the quasigroup completion problem. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, pages 288–302, 2003.

[14] K. Easton, G.L. Nemhauser, and M.A. Trick. Solving the travelling tournament problem: A combined integer programming and constraint programming approach. In *Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling IV*, pages 100–112, 2002.

[15] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 97–104, 1993.

[16] M.S. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 1983.

[17] M.S. Fox, B. Allen, and G. Strohm. Job-shop scheduling: An investigation in constraint-directed reasoning. In *Proceedings of the 2nd Conference of The American Association for Artificial Intelligence*, pages 155–158, 1982.

[18] E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1-3):21–70, 1992.

[19] I.P. Gent and T. Walsh. CSPLib: A benchmark library for constraints. In *Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming*, pages 480–481, 1999. Available at `http://www.csplib.org/`.

[20] P. Van Hentenryck and L. Michel. Nondeterministic control for hybrid search. In *Proceedings of the 2nd International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 380–395, 2005.

[21] P. Van Hentenryck and L. Michel. Nondeterministic control for hybrid search. *Constraints*, 11(4):353–373, 2006.

[22] B. Hnich, B. Smith, and T. Walsh. Dual modelling of permutation and injection problems. *Journal of Artificial Intelligence Research*, 21:357–391, 2004.

[23] B. Hnich and T. Walsh. Models of injection problems. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, page 781, 2002.

[24] J.N. Hooker, G. Ottosson, E.S. Thorsteinsson, and H.J. Kim. On integrating constraint propagation and linear programming for combinatorial optimization. In *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 136–141, 1999.

[25] A.H. Land and A.G. Doig. An automatic method for solving discrete programming problems. *Eco nometrica*, 28:497–520, 1960.

[26] J. Larrosa. Node and arc consistency in weighted CSP. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 48–53, 2002.

[27] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 239–244, 2003.

[28] J. Larrosa and T. Schiex. Solving weighted CSP by maintaining arc consistency. *Artificial Intelligence*, 159(1-2):1–26, 2004.

[29] 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.

[30] Y.C. Law, J.H.M. Lee, and B.M. Smith. Automatic generation of redundant models for permutation constraint satisfaction problems. *Constraints*, 12(4):469–505, 2007.

[31] Y.C. Law, J.H.M. Lee, and M.H.C. Woo. Speeding up weighted constraint satisfaction using redundant modeling. In *Proceedings of the 19th Australian Joint Conference on Artificial Intelligence*, pages 59–68, 2006.

[32] Y.C. Law, J.H.M. Lee, and M.H.C. Woo. A parameterized local consistency for redundant modeling in weighted csps. In *Proceedings of the 20th Australian Joint Conference on Artificial Intelligence*, pages 191–201, 2007.

[33] J.H.M. Lee and C.F.K. Siu. Weighted constraint satisfaction with set variables. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 80–85, 2006.

[34] J.H.M. Lee and C.F.K. Siu. Stronger consistencies in wcsps with set variables. In *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence*, pages 291–298, 2008.

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

[36] E. Manisterski, D. Sarne, and S. Kraus. Cooperative search with concurrent interactions. *Journal of Artificial Intelligence Research*, 32:1–36, 2008.

[37] P. Marti and M. Rueher. A distributed cooperating constraints solving system. *International Journal on Artificial Intelligence Tools*, 4:4–1, 1995.

[38] A. Montoyo, A. Suarez, G. Rigau, and M. Palomar. Combining knowledge- and corpus-based word-sense-disambiguation methods. *Journal of Artificial Intelligence Research*, 23:299–330, 2005.

[39] B.A. Nadel. Representation selection for constraint satisfaction: A case study using n-queens. *IEEE Expert: Intelligent Systems and Their Applications*, 5(3):16–23, 1990.

[40] Z. Ruttkay. Fuzzy constraint satisfaction. In *Proceedings of the 1st IEEE Conference on Evolutionary Computing*, pages 542–547, 1994.

[41] A. Sathi and M.S. Fox. Constraint-directed negotiation of resource reallocations. In *Distributed Artificial Intelligence (Vol. 2)*, pages 163–193, 1989.

[42] T. Schiex. Possibilistic constraint satisfaction problems or "how to handle soft constraints?". In *Proceedings of the 8th conference on Uncertainty in Artificial Intelligence*, pages 268–275, 1992.

[43] T. Schiex. Arc consistency for soft constraints. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming*, pages 411–424, 2000.

[44] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: hard and easy problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 631–637, 1995.

[45] L.G. Shapiro and R.M. Haralick. Structural descriptions and inexact matching. *IEEE Transactions Pattern Analysis Machine Intelligence*, 3:504–519, 1981.

[46] B.M. Smith. Modelling a permutation problem. In *Proceedings of ECAI'2000 Workshop on Modelling and Solving Problems with Constraints*, 2000.

[47] B.M. Smith. Dual models of permutation problems. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pages 615–619, 2001.

[48] M. Wallace. Hybrid algorithms in constraint programming. In *Proceedings of the 11th Annual ERCIM International Workshop on Constraint Solving and Contraint Logic Programming*, pages 1–32, 2006.

[49] T. Walsh. Permutation Problems and Channelling Constraints. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, pages 377–391, 2001.

[50] D. Waltz. Understanding line drawings of scenes with shadows. In *The Psychology of Computer Vision*, pages 19–91, 1975.