Speeding Up Weighted Constraint Satisfaction Using Redundant Modeling*

Y.C. Law, J.H.M. Lee, and 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, combining mutually redundant models using channeling constraints is effective in increasing constraint propagation and reducing search space for many problems. In this paper, we investigate how to benefit the same for *weighted constraint satisfaction problems* (WCSPs), a common soft constraint framework for modeling optimization and over-constrained problems. First, we show how to generate a redundant WCSP model from an existing WCSP using *generalized model induction*. We then uncover why naively combining two WCSPs by posting channeling constraints as hard constraints and relying on the standard NC* and AC* propagation algorithms does not work well. Based on these observations, we propose m-NC$^*_c$ and m-AC$^*_c$ and their associated algorithms for effectively enforcing node and arc consistencies on a combined model with m sub-models. The two notions are strictly stronger than NC* and AC* respectively. Experimental results confirm that applying the 2-NC$^*_c$ and 2-AC$^*_c$ algorithms on combined models reduces more search space and runtime than applying the state-of-the-art AC*, FDAC*, and EDAC* algorithms on single models.

1 Introduction

Redundant modeling [1] has been shown effective in increasing constraint propagation and solving efficiency for *constraint satisfaction problems* (CSPs). While the technique, which is to combine two different CSP models of a problem using *channeling constraints*, is applied successfully to classical CSPs, we study in this paper how to benefit the same for *weighted CSPs* (WCSPs) [2, 3], a common soft constraint framework for modeling optimization and over-constrained problems.

Unlike classical CSPs, obtaining mutually redundant WCSP models for a problem is more difficult in general, since each problem solution is associated with a cost. Besides the problem requirements, we need to ensure the same cost distribution on the solutions of the two models. While model induction [4] can automatically generate a redundant classical CSP from a given one, we *generalize* the notion so that redundant permutation WCSPs can also be generated.

* We thank the anonymous referees for their constructive comments. The work described in this paper was substantially supported by grants (CUHK4358/02E and CUHK4219/04E) from the Research Grants Council of Hong Kong SAR.

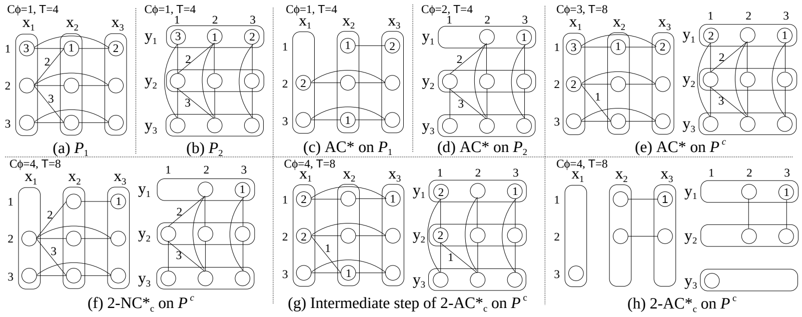
For classical CSPs, we can rely on the standard propagation algorithms of the channeling constraints to transmit pruning information between sub-models to achieve stronger propagation. We can also do the same to combine WCSPs by posting the channeling constraints as hard constraints. However, we discover that applying the standard AC* algorithm [3] to the combined model results in weaker constraint propagation and worse performance. Some prunings that are available when propagating a single model alone would even be missed in a combined model. Based on these observations, we *generalize* the notions of node and arc consistencies and *propose* m-NC$^*_\mathrm{c}$ and m-AC$^*_\mathrm{c}$ for a combined model with m sub-models. The m-NC$^*_\mathrm{c}$ (*resp.* m-AC$^*_\mathrm{c}$) has strictly stronger propagation behavior than NC* (*resp.* AC*) and degenerates to NC* (*resp.* AC*) when $m = 1$. While m-NC$^*_\mathrm{c}$ and m-AC$^*_\mathrm{c}$ are applicable to general WCSPs, we focus our discussions on permutation WCSPs. Experimental results confirm that 2-AC$^*_\mathrm{c}$ is particularly useful to solve hard problem instances. Enforcing 2-AC$^*_\mathrm{c}$ on combined models reduces significantly more search space and runtime than enforcing the state-of-the-art local consistencies AC* [3], FDAC* [5], and EDAC* [6] on single models.

2 Background

WCSPs associate *costs* to tuples [7]. The costs are specified by a *valuation structure* $S(k) = ([0, \ldots, k], \oplus, \geq)$, where $k \in \{1, \ldots, \infty\}$, $\oplus$ is defined as $a \oplus b = \min\{k, a + b\}$, and $\geq$ is the standard order among naturals. The minimum and maximum costs are denoted by $\bot = 0$ and $\top = k$ 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 $x_i \in \mathcal{X}$. An *assignment* $x \mapsto a$ in $\mathcal{P}$ is a mapping from variable x to value $a \in D_x$. 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*. A unary constraint involving variable x is a cost function $C_x : D_x \to \{0, \ldots, k\}$. A binary constraint involving variables x and y is a cost function $C_{x,y} : D_x \times D_y \to \{0, \ldots, k\}$. We also assume a *zero-arity* constraint $C_\emptyset$ in $\mathcal{P}$ which is a constant denoting the global lower bound of costs in $\mathcal{P}$. Fig. 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 labeled edges connecting two assignments. Unlabeled edges have $\top$ cost; $\bot$ costs are not shown for clarity.

The cost of a complete tuple $\theta = \{x_i \mapsto v_i \,|\, 1 \leq i \leq n\}$ in $\mathcal{P}$ is $\mathcal{V}(\theta) = C_\emptyset \oplus \sum_i C_{x_i}(v_i) \oplus \sum_{i<j} C_{x_i,x_j}(v_i, v_j)$. If $\mathcal{V}(\theta) < \top$, θ is a *solution* of $\mathcal{P}$. Solving a WCSP is to find a solution θ with minimized $\mathcal{V}(\theta)$, which is NP-hard. The WCSP in Fig. 1(a) has a minimum cost 2 with the solution $\{x_1 \mapsto 3, x_2 \mapsto 1, x_3 \mapsto 2\}$ $(C_\emptyset \oplus C_{x_1}(3) \oplus C_{x_2}(1) \oplus C_{x_3}(2) \oplus C_{x_1,x_2}(3, 1) \oplus C_{x_1,x_3}(3, 2) \oplus C_{x_2,x_3}(1, 2) = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0 = 2)$. A WCSP is equivalent to a classical CSP if each cost in the WCSP is either $\bot$ or $\top$. 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.

Following Schiex [2] and Larrosa [3], we define node and arc consistencies for WCSPs as follows. They degenerate to their standard counterparts for CSPs.



(a) P_1 (b) P_2 (c) AC* on P_1 (d) AC* on P_2 (e) AC* on P^c (f) 2-NC*$_c$ on P^c (g) Intermediate step of 2-AC*$_c$ on P^c (h) 2-AC*$_c$ on P^c

Fig. 1. Enforcing node and arc consistencies on WCSPs $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}^c$

Definition 1. *Let $\mathcal{P} = (k, \mathcal{X}, \mathcal{D}, \mathcal{C})$ 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 $x \mapsto a$ in $\mathcal{P}$ is* arc consistent *(AC) with respect to a constraint $C_{x,y}$ if there exists an assignment $y \mapsto b$ of y such that $C_{x,y}(a, b) = \bot$. Value b is a* support *for $x \mapsto a$. A variable x in $\mathcal{P}$ is AC if all assignments of x are AC with respect to all binary constraints involving x. $\mathcal{P}$ is* star arc consistent *(AC*) if every variable in $\mathcal{P}$ is NC* and AC.*

NC* and AC* are enforced by forcing supports for variables and pruning node inconsistent values. Supports can be forced by *projections* of unary (*resp.* binary) constraints over $C_\emptyset$ (*resp.* unary constraints) [2,3]. Let $0 \leq b \leq a \leq k$, we define *subtraction* as $a \ominus b = a - b$ if $a \neq k$; and $a \ominus b = k$ otherwise. Let $\alpha = \min_{a \in D_x}\{C_x(a)\}$ for a variable x. *Projection* of C_x over $C_\emptyset$ [3] is defined such that $C_\emptyset := C_\emptyset \oplus \alpha$ and for each $a \in D_x$, $C_x(a) := C_x(a) \ominus \alpha$. After forcing supports for all variables, all assignments $x \mapsto a$ with $C_\emptyset \oplus C_x(a) = \top$ can be removed. NC* can be enforced on a WCSP with n variables and maximum domain size d in $O(nd)$ time [3]. The WCSP in Fig. 1(a) is not NC*. Removing value 1 from D_{x_1} makes it NC*. Similarly, given variables x and y, for each $a \in D_x$, let $\alpha_a = \min_{b \in D_y}\{C_{x,y}(a, b)\}$. *Projection* of $C_{x,y}$ over C_x [2,3] is defined such that for each $a \in D_x$, $C_x(a) := C_x(a) \oplus \alpha_a$ and for each $a \in D_x$ and $b \in D_y$, $C_{x,y}(a, b) := C_{x,y}(a, b) \ominus \alpha_a$. Consider the assignment $x_1 \mapsto 2$ and the constraint C_{x_1,x_2} in Fig. 1(a), 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-$\bot$. We can subtract 2 from each of these costs and add 2 to $C_{x_1}(2)$ to force a support for $x_1 \mapsto 2$. AC* can be enforced using a fix point algorithm in $O(n^2d^3)$ time [3]. The WCSP in Fig. 1(c) is AC* and equivalent to the one in Fig. 1(a).

3 Generating Redundant WCSP Models

Deriving multiple classical CSP models for the same problem is common, although not trivial. Cheng et al. [1] modeled a real-life nurse rostering problem as two different CSPs and showed that combining those CSPs using *channeling constraints* can increase constraint propagation. Hnich et al. [8] made an extensive study of combining different models for permutation and injection problems. Law and Lee [4] proposed *model induction* which automatically generates a redundant CSP from a given one. Two CSPs $\mathcal{P}_1$ and $\mathcal{P}_2$ are *mutually redundant* if there is a bijection between the two sets of all solutions of $\mathcal{P}_1$ and $\mathcal{P}_2$. For two WCSPs $\mathcal{P}_1$ and $\mathcal{P}_2$, we further require that if a solution θ_1 of $\mathcal{P}_1$ corresponds to a solution θ_2 of $\mathcal{P}_2$, then $\mathcal{V}(\theta_1) = \mathcal{V}(\theta_2)$. Thus, deriving mutually redundant WCSP models is more difficult. We propose here a slight generalization of model induction that generates mutually redundant *permutation* WCSPs.

A *permutation WCSP* (PermWCSP) is a WCSP in which all variables have the same domain, the number of variables equals the domain size, and every solution assigns a permutation of the domain values to the variables, i.e., we can set the costs $C_{x_i,x_j}(a,a) = \top$ for all variables x_i and x_j and domain value a. Given a PermWCSP $\mathcal{P} = (k, \mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X})$, we can always interchange the roles of its variables and values to give a *dual* PermWCSP. If $\mathcal{X} = \{x_1, \ldots, x_n\}$ and $D_{x_i} = \{1, \ldots, n\}$, a dual model is $\mathcal{P}' = (k, \mathcal{Y}, \mathcal{D}_\mathcal{Y}, \mathcal{C}_\mathcal{Y})$ with $\mathcal{Y} = \{y_1, \ldots, y_n\}$ and $D_{y_j} = \{1, \ldots, n\}$. The relationship between the variables in $\mathcal{X}$ and $\mathcal{Y}$ can be expressed using the channeling constraints $x_i = j \Leftrightarrow y_j = i$ for $1 \le i, j \le n$.

Generalized model induction of a WCSP $\mathcal{P}$ requires a *channel function* that maps assignments in $\mathcal{P}$ to those in another set of variables. If $\mathcal{P}$ is a PermWCSP, 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* are defined such that for $1 \le a, i \le n$, $C_{y_a}(i) = C_{x_i}(a)$, and for $1 \le a, b, i, j \le n$, $C_{y_a,y_b}(i,j) = C_{x_i,x_j}(a,b)$ if $i \ne j$; and $C_{y_a,y_b}(i,j) = \top$ otherwise. Note that the induced model must be a PermWCSP, since $C_{y_a,y_b}(i,i) = \top$ for all $1 \le a, b, i \le n$. Fig. 1(b) shows the induced model $\mathcal{P}_2$ of $\mathcal{P}_1$ in Fig. 1(a). In the example, we have, say, the unary cost $C_{y_1}(2) = C_{x_2}(1) = 1$ and the binary cost $C_{y_2,y_3}(1,2) = C_{x_1,x_2}(2,3) = 3$.

4 Combining Mutually Redundant WCSPs

Following the redundant modeling [1] technique, we can also combine two mutually redundant WCSPs $\mathcal{P}_1 = (k_1, \mathcal{X}, \mathcal{D}_\mathcal{X}, \mathcal{C}_\mathcal{X})$ and $\mathcal{P}_2 = (k_2, \mathcal{Y}, \mathcal{D}_\mathcal{Y}, \mathcal{C}_\mathcal{Y})$ using a set of channeling constraints $\mathcal{C}^c$ to give the *combined model* $\mathcal{P}^c = (k_1 + k_2, \mathcal{X} \cup \mathcal{Y}, \mathcal{D}_\mathcal{X} \cup \mathcal{D}_\mathcal{Y}, \mathcal{C}_\mathcal{X} \cup \mathcal{C}_\mathcal{Y} \cup \mathcal{C}^c)$. In $\mathcal{P}^c$, $\mathcal{C}^c$ contains hard constraints. For example, the channeling constraint $x_1 = 2 \Leftrightarrow y_2 = 1$ has the cost function $C_{x_1,y_2}(a,b) = \bot$ if $a = 2 \Leftrightarrow b = 1$; and $C_{x_1,y_2}(a,b) = \top$ otherwise. We perform some preliminary experiments in ToolBar,[1] a branch and bound WCSP solver maintaining local consistencies at each search node, using Langford's problem (prob024 in CSPLib[2]) to evaluate the performance of the single and combined models. The single model $\mathcal{P}$ used is based on a model by Hnich et al. [8] Since the problem is over-constrained for many instances, we soften $\mathcal{P}$ so that the con-

[1] Available at `http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro`.
[2] Available at `http://www.csplib.org`.

straints except the all-different constraint can have random non-$\top$ costs. The combined model $\mathcal{P}^c$ contains $\mathcal{P}$ and its induced model as sub-models.

Unlike redundant modeling in classical CSPs, enforcing AC* on $\mathcal{P}^c$ is even weaker than on $\mathcal{P}$. The former requires more backtracks than the latter in the search, and solving $\mathcal{P}^c$ is about three times slower than solving $\mathcal{P}$. This is mainly due to three reasons. First, there are more variables and constraints in $\mathcal{P}^c$ than in $\mathcal{P}$. It takes longer time to propagate the constraints. Second, we need a large number of channeling constraints to connect two models. For CSPs, efficient global constraints exist for propagating the channeling constraints, but there are no such counterparts for WCSPs. Third, we discover that enforcing AC* on $\mathcal{P}^c$ can miss some prunings which are available even in $\mathcal{P}$. For example, Figs. 1(a) and 1(b) show two mutually redundant WCSPs $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, which can be combined using $\mathcal{C}^c = \{x_i = j \Leftrightarrow y_j = i \,|\, 1 \leq i, j \leq 3\}$ to give $\mathcal{P}^c$ with the valuation structure $S(4+4) = S(8)$ and $C_\emptyset = 1 \oplus 1 = 2$. Figs. 1(c), 1(d), and 1(e) respectively show the AC* equivalent of $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}^c$. Note that $x_1 \mapsto 1$ and $y_1 \mapsto 1$ are removed in Figs. 1(c) and 1(d) respectively. However, in Fig. 1(e), the assignments are still NC* and AC*, since $C_\emptyset \oplus C_{x_1}(1) = 3 \oplus 3 < \top = 8$ and $C_\emptyset \oplus C_{y_1}(1) = 3 \oplus 2 < \top = 8$, and thus they cannot be removed. This example shows the undesirable behavior that enforcing AC* on a combined model results in weaker constraint propagation than on its sub-models individually.

To remedy the drawbacks, in the following subsections, 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 with no channeling constraints.

4.1 Node Consistency Revisited

We observe in the above example that given any solution θ of $\mathcal{P}^c$, if an assignment $x_i \mapsto j$ in $\mathcal{P}_1$ is in θ, then according to the channeling constraints, the corresponding assignment $y_j \mapsto i$ in $\mathcal{P}_2$ must be also in θ, and vice versa. Therefore, we can check the node consistencies of $x_i \mapsto j$ and $y_j \mapsto i$ simultaneously. If $C_\emptyset \oplus C_{x_i}(j) \oplus C_{y_j}(i) = \top$, then both $x_i \mapsto j$ and $y_j \mapsto i$ cannot be in any solution of $\mathcal{P}^c$ and can be pruned. Consider the assignments $x_1 \mapsto 1$ and $y_1 \mapsto 1$ in $\mathcal{P}^c$ in Fig. 1(e), since $C_\emptyset \oplus C_{x_1}(1) \oplus C_{y_1}(1) = 3 \oplus 3 \oplus 2 = 8 = \top$, both assignments should be pruned, thus restoring the available prunings in the single models.

Furthermore, a complete tuple of $\mathcal{P}_2$ must contain exactly one assignment of, say, y_1. The set of assignments $\{y_1 \mapsto 1, y_1 \mapsto 2, y_1 \mapsto 3\}$ in $\mathcal{P}_2$ corresponds 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 in θ. In Fig. 1(e), 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 $C_\emptyset$ of $\mathcal{P}^c$.

By capturing the previously described ideas, 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 PermWCSPs only. In the following, we assume $\mathcal{P}_s = (k_s, \mathcal{X}_s, \mathcal{D}_s, \mathcal{C}_s)$ for $1 \leq s \leq m$ are m mutually redundant WCSPs, where $\mathcal{X}_s = \{x_{s,i} \,|\, 1 \leq i \leq n_s\}$ and $\mathcal{D}_s = \{D_{x_{s,i}} \,|\, 1 \leq i \leq n_s\}$

($n_s = |\mathcal{X}_s|$). $\mathcal{C}_{s,t}$ is the set of channeling constraints connecting $\mathcal{P}_s$ and $\mathcal{P}_t$, and $\mathcal{C}^c = \bigcup_{s<t} \mathcal{C}_{s,t}$. $\mathcal{P}^c = (k, \mathcal{X}, \mathcal{D}, \mathcal{C})$ is a combined model of all m sub-models $\mathcal{P}_s$, where $k = \sum_s k_s$, $\mathcal{X} = \bigcup_s \mathcal{X}_s$, $\mathcal{D} = \bigcup_s \mathcal{D}_s$, and $\mathcal{C} = \bigcup_s \mathcal{C}_s \cup \mathcal{C}^c$. 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 2. *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$_\mathrm{c}^*$ *if every variable in $\mathcal{X}$ is m-NC$_\mathrm{c}^*$.*

For example, $\mathcal{P}^c$ in Fig. 1(e) is NC* (and AC*) but *not* 2-NC$_\mathrm{c}^*$, since (1) $C_\emptyset \oplus C_{x_1}(1) \oplus C_{y_1}(1) = \top$ and (2) there are no c-supports for the tuple $\{x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 1\}$. Fig. 1(f) shows its 2-NC$_\mathrm{c}^*$ equivalent. Note that 1-NC$_\mathrm{c}^*$ is equivalent to NC*, while m-NC$_\mathrm{c}^*$ is a stronger notion of consistency than NC*.

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

To enforce m-NC$_\mathrm{c}^*$ on $\mathcal{P}^c$, we propose a new form of projection which forces c-supports for tuples. Given a tuple θ, let $\alpha = \min_{(x \mapsto a) \in \theta}\{C_x(a)\}$. *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 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 $C_x(a) = \bot$. Given a variable $x_{s,i}$ in $\mathcal{P}^c$, c-projection of $\vartheta_t(x_{s,i})$ over $C_\emptyset$ transforms $\mathcal{P}^c$ into an equivalent WCSP.

In the above example, $\theta = \{x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 1\}$ is the set of assignments in $\mathcal{P}_1$ corresponding to the set of all assignments of y_1 in $\mathcal{P}_2$. Hence, c-projection of θ over $C_\emptyset$ maintains the same cost distribution on complete tuples. In the original $\mathcal{P}^c$, $C_{x_1}(1) = 3$, $C_{x_2}(1) = 1$, and $C_{x_3}(1) = 2$, c-projection of θ deducts 1 from each of the costs and increases $C_\emptyset$ by 1, forcing a c-support $x_2 \mapsto 1$ for θ.

Fig. 2(a) shows an algorithm for enforcing m-NC$_\mathrm{c}^*$ on a combined model $\mathcal{P}^c$. The algorithm first forces a c-support for each $\vartheta_t(x_{s,i})$ by c-projecting each $\vartheta_t(x_{s,i})$ over $C_\emptyset$. Next, for each $x_{s,i} \in \mathcal{X}$, $\mathsf{pruneVar_c}(x_{s,i})$ is called to prune any non-m-NC$_\mathrm{c}^*$ assignments. By using table lookup, a channel function can be implemented in $O(1)$ time. Therefore, $\mathsf{pruneVar_c}()$ and $\mathsf{NC_c^*}()$ runs in $O(md)$ and $O(mnd)$ time respectively, where d is the maximum domain size and $n = |\mathcal{X}|$.

In $\mathcal{P}^c$, when an assignment $x_{s,i} \mapsto a$ is not m-NC$_\mathrm{c}^*$, all $f_{s,t}(x_{s,i} \mapsto a)$ 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 done all the propagation of the channeling constraints, and

```
procedure NC*_c(k, 𝒳, 𝒟, 𝒞)
for each x_{s,i} ∈ 𝒳 each 1 ≤ t ≤ m do
  α := min_{(x_{t,j} ↦ b) ∈ ϑ_t(x_{s,i})} {C_{x_{t,j}}(b)};
  C_∅ := C_∅ ⊕ α;
  for each (x_{t,j} ↦ b) ∈ ϑ_t(x_{s,i}) do
    C_{x_{t,j}}(b) := C_{x_{t,j}}(b) ⊖ α;
for each x_{s,i} ∈ 𝒳 do
  pruneVar_c(x_{s,i});
endprocedure
procedure pruneVar_c(x_{s,i})
for each a ∈ D_{x_{s,i}} do
  let x_{t,j} ↦ b_t ≡ f_{s,t}(x_{s,i} ↦ a) for 1 ≤ t ≤ m
  if C_∅ ⊕ Σ_t C_{x_{t,j}}(b_t) = ⊤ then
    for each 1 ≤ t ≤ m do
      D_{x_{t,j}} := D_{x_{t,j}} \ {b_t};
endprocedure
```

(a) m-NC$^*_\mathrm{c}$ algorithm

```
procedure findCSupport(x_{s,i}, x_{s,j})
for each 1 ≤ t ≤ m do
  supported := true;
  for each (x_{t,i'} ↦ a') ∈ ϑ_t(x_{s,i}) do
    if S(x_{t,i'} ↦ a', x_{s,j}, t) ∉ ϑ_t(x_{s,j}) then
      let x_{t,j*} ↦ b* ≡ argmin_{(x_{t,j'} ↦ b') ∈ ϑ_t(x_{s,j})} {C_{x_{t,i'}, x_{t,j'}}(a', b')};
      S(x_{t,i'} ↦ a', x_{s,j}, t) := (x_{t,j*} ↦ b*);
      α := C_{x_{t,i'}, x_{t,j*}}(a', b*);    C_{x_{t,i'}}(a') := C_{x_{t,i'}}(a') ⊕ α;
      for each (x_{t,j'} ↦ b') ∈ ϑ_t(x_{t,j}) do
        C_{x_{t,i'}, x_{t,j'}}(a', b') := C_{x_{t,i'}, x_{t,j'}}(a', b') ⊖ α;
      if C_{x_{t,i'}}(a') = ⊥ ∧ α > ⊥ then supported := false;
  if ¬supported then
    let x_{t,i*} ↦ a* ≡ argmin_{(x_{t,i'} ↦ a') ∈ ϑ_t(x_{s,i})} {C_{x_{t,i'}}(a')};
    S(x_{s,i}, t) := (x_{t,i*} ↦ a*);
    α := C_{x_{t,i*}}(a*);    C_∅ := C_∅ ⊕ α;
    for each (x_{t,i'} ↦ a') ∈ ϑ_t(x_{s,i}) do
      C_{x_{t,i'}}(a') := C_{x_{t,i'}}(a') ⊖ α;
endprocedure
```

(b) m-AC$^*_\mathrm{c}$ algorithm

Fig. 2. Algorithms for enforcing m-NC$^*_\mathrm{c}$ and m-AC$^*_\mathrm{c}$

we can skip posting them in $\mathcal{P}^c$ to save propagation overhead. In subsequent discussions, we assume no channeling constraints exist in a combined model if m-NC$^*_\mathrm{c}$ is enforced.

4.2 Arc Consistency Revisited

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 $\mathcal{P}_2$. Now for the assignment $y_2 \mapsto 1$ in Fig. 1(e), there is a binary cost $C_{y_2,y_j}(1,2)$ incurred between $y_2 \mapsto 1$ and $(y_j \mapsto 2) \in \theta$. (When $j = 2$, there are actually no such cost, but we assume without losing generality that such "cost" is $\top$.) Since the minimum cost among $C_{y_2,y_1}(1,2) = 2$, "$C_{y_2,y_2}(1,2)$" $= \top$, and $C_{y_2,y_3}(1,2) = 3$ is $2 > \bot$, we can use such information to tighten the bound on the cost $C_{y_2}(1)$. Thus, we can propose a new arc consistency m-AC$^*_\mathrm{c}$ for combined models with m sub-models.

Definition 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 $C_{x_{t,i'},x_{t,j'}}(a', b') = \bot$, 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$^*_\mathrm{c}$.*

The combined model $\mathcal{P}^c$ in Fig. 1(e) is AC* but *not* 2-AC$^*_\mathrm{c}$, since there are no c-supports among $\{y_1 \mapsto 2, y_2 \mapsto 2, y_3 \mapsto 2\}$ for $y_2 \mapsto 1$. Fig. 1(h) shows an equivalent 2-AC$^*_\mathrm{c}$ WCSP. Again, 1-AC$^*_\mathrm{c}$ is equivalent to AC*, while m-AC$^*_\mathrm{c}$ is a stronger notion of consistency than AC*.

Theorem 2. *Let $\mathcal{P}^c$ be a combined model of m sub-models $\mathcal{P}_s$ for $1 \leq s \leq m$. Enforcing m-AC$^*_\mathrm{c}$ on $\mathcal{P}^c$ is strictly stronger than enforcing AC* on $\mathcal{P}^c$.*

To enforce m-AC$_c^*$ on a combined model, we extend the definition of c-projections which can force c-supports for assignments. Given a tuple θ and an assignment $(x \mapsto a) \notin \theta$, let $\alpha = \min_{(y \mapsto b) \in \theta}\{C_{x,y}(a,b)\}$. *C-projection* of 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 $(y \mapsto b) \in \theta$, $C_{x,y}(a,b) := C_{x,y}(a,b) \ominus \alpha$. C-projection of the set of all assignments of a variable y over $x \mapsto a$ is equivalent to ordinary binary projection. Given an assignment $(x_{t,j} \mapsto a) \notin \vartheta_t(x_{s,i})$ in a combined model $\mathcal{P}^c$, c-projection of $\vartheta_t(x_{s,i})$ over $x_{t,j} \mapsto a$ transforms $\mathcal{P}^c$ into an equivalent WCSP.

Recall the combined model $\mathcal{P}^c$ in Fig. 1(e), $y_2 \mapsto 1$ has no c-supports in $\theta = \{y_1 \mapsto 2, y_2 \mapsto 2, y_3 \mapsto 2\}$, which is the corresponding set of all assignments of x_2 in $\mathcal{P}_1$. C-projections of θ over $y_2 \mapsto 1$ yields $C_{y_2,y_1}(1,2) = \bot$, $C_{y_2,y_3}(1,2) = 1$, and $C_{y_2}(1) = 2$, as shown in Fig. 1(g). In the figure, $\{x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 1\}$ is also c-projected over $C_\emptyset$ such that $C_{x_1}(1) = 2$, $C_{x_2}(1) = \bot$, $C_{x_3}(1) = 1$, and $C_\emptyset = 4$. The assignments $x_1 \mapsto 1$, $x_1 \mapsto 2$, $y_1 \mapsto 1$, and $y_2 \mapsto 1$ are consequently not 2-NC$_c^*$, since $C_\emptyset \oplus C_{x_1}(1) \oplus C_{y_1}(1) = C_\emptyset \oplus C_{x_1}(2) \oplus C_{y_2}(1) = 4 \oplus 2 \oplus 2 = \top$, and are thus pruned. Variable x_1 is now bound and further propagation yields the 2-AC$_c^*$ WCSP in Fig. 1(h). The optimal solution is $\{x_1 \mapsto 3, x_2 \mapsto 1, x_3 \mapsto 2, y_1 \mapsto 2, y_2 \mapsto 3, y_3 \mapsto 1\}$, which has aggregate cost 4.

Fig. 2(b) shows the core algorithm for enforcing m-AC$_c^*$. It uses two data structures $S(x_{t,i'} \mapsto a', x_{s,j}, t)$ and $S(x_{s,i}, t)$. They store the current c-support for the assignment $x_{t,i'} \mapsto a'$ among $\vartheta_t(x_{s,j})$ and for $\vartheta_t(x_{s,i})$ respectively. The algorithm generalizes the procedure findSupport() by Larrosa [3] so that for each sub-model $\mathcal{P}_t$, it forces a c-support among $\vartheta_t(x_{s,j})$ for each assignment $(x_{t,i'} \mapsto a') \in \vartheta_t(x_{s,i})$ (lines 3–10). C-projections over $C_\emptyset$ are done when necessary (lines 11–16). After finding c-supports, any assignments that are not m-NC$_c^*$ are pruned using pruneVar$_c$() in Fig. 2(a). The findCSupport() algorithm runs in $O(md^2)$ time, hence the overall m-AC$_c^*$ algorithm runs in $O(mn^2d^3)$ time.

5 Experiments

We implement the 2-NC$_c^*$ and 2-AC$_c^*$ algorithms in ToolBar to evaluate their efficiency on combined models, using Langford's problem and Latin square problem, both of which can be modeled as PermWCSPs. Enforcing AC* on combined models is highly inefficient, so comparisons are made among AC* [3], FDAC* [5], and EDAC* [6] on a single model $\mathcal{P}$ and 2-AC$_c^*$ on a combined model $\mathcal{P}^c$, which contains $\mathcal{P}$ and its induced model as sub-models. Experiments are run on a 1.6GHz US-IIIi CPU with 2GB memory. We use the *dom/deg* variable ordering heuristic [6] and the smallest-cost-first value ordering heuristic [6]. The initial $\top$ provided to ToolBar is n^2, where n is the number of variables in a single model. We report in the tables the average number of fails (i.e., the number of backtracks occurred in solving a model) and CPU time in seconds to find an optimal solution. The first column shows the problem instances; those marked with "*" have a $\bot$ optimal cost. The subsequent columns show the results of enforcing various local consistencies on either $\mathcal{P}$ or $\mathcal{P}^c$. A cell labeled with "-" denotes a timeout after 2 hours.

Table 1. Experimental results on solving Langford's problem and Latin square problem

(m,n)	(a) Classical Langford's problem								(b) Soft Langford's problem							
	AC* on $\mathcal{P}$		FDAC* on $\mathcal{P}$		EDAC* on $\mathcal{P}$		2-AC$_c^*$ on $\mathcal{P}^c$		AC* on $\mathcal{P}$		FDAC* on $\mathcal{P}$		EDAC* on $\mathcal{P}$		2-AC$_c^*$ on $\mathcal{P}^c$	
	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time	fail	time
(2, 6)	80	0.01	80	**0**	80	0.01	**57**	0.01	180	**0.01**	177	0.02	176	0.02	**130**	0.02
(2, 7)*	**1**	**0**	**1**	0.01	**1**	0.01	2	0.01	91	**0.01**	**79**	**0.01**	80	**0.01**	90	**0.01**
(2, 8)*	22	0.01	18	**0**	18	**0**	**0**	0.01	162	**0.01**	146	**0.01**	145	0.02	**137**	0.03
(2, 9)	8576	0.85	8576	0.8	8576	1	**4209**	**0.79**	10850	1.09	10823	1.05	10823	1.05	**5074**	**1**
(2, 10)	48048	5.03	48048	4.91	48048	6.07	**22378**	**4.48**	59681	6.38	59654	6.14	59655	6.13	**25718**	**5.42**
(2, 11)*	11	**0.01**	8	**0.01**	11	**0.01**	**3**	0.02	595	**0.07**	547	**0.07**	546	0.08	**405**	0.1
(2, 12)*	7	**0.01**	7	**0.01**	7	**0.01**	**0**	0.02	795	**0.1**	708	**0.1**	700	0.12	**576**	0.17
(2, 13)	14.73M	1769.8	14.73M	1691.98	14.73M	2164.02	**5.72M**	**1325.92**	18.16M	2232.01	18.16M	2142.04	18.16M	2160.93	**6.46M**	**1552.65**
(3, 5)	6	**0**	6	**0**	6	**0**	**4**	0.01	368	**0.04**	285	**0.04**	**279**	0.05	284	0.06
(3, 6)	20	**0.01**	20	**0.01**	20	**0.01**	**14**	0.02	901	0.14	681	**0.12**	668	0.15	**600**	0.19
(3, 7)	62	0.04	62	**0.03**	62	0.04	**29**	0.04	2286	0.51	1687	**0.42**	1687	0.53	**1559**	0.67
(3, 8)	238	0.13	238	**0.1**	238	0.12	**96**	0.13	2735	0.81	1976	**0.61**	**1962**	0.75	2398	1.23
(3, 9)*	195	0.12	193	**0.09**	192	0.11	**53**	0.11	2665	0.53	**1169**	**0.31**	1360	0.44	3893	1.94
(3, 10)*	590	0.42	560	0.29	560	0.39	**97**	**0.23**	6612	1.74	4026	**1.34**	**3903**	1.6	8715	5.2
(3, 11)	14512	10.01	14512	7.45	14512	10.09	**3029**	**6.15**	77507	43.8	69356	33.89	69455	42.3	**27992**	**33.27**
(3, 12)	62016	46.13	62016	35.19	62016	46.69	**12252**	**25.57**	275643	178.25	252527	136.27	252830	171.14	**82804**	**119.44**
(3, 13)	300800	247.45	300800	191.56	300800	257.59	**45274**	**108.71**	949361	736.13	920007	577.57	919749	720.18	**174881**	**354.39**
(3, 14)	1.37M	1185.08	1.37M	933.22	1.37M	1229.83	**190153**	**525.88**	4.41M	3704.64	4.32M	2886.43	4.32M	3624	**596201**	**1448.54**
(3, 15)	7.52M	6992.09	7.52M	5419.17	-	-	**851968**	**2750.05**	-	-	-	-	-	-	**2.31M**	**6742.25**

n	(c) Classical Latin square problem							
	AC* on $\mathcal{P}$		FDAC* on $\mathcal{P}$		EDAC* on $\mathcal{P}$		2-AC$_c^*$ on $\mathcal{P}^c$	
	fail	time	fail	time	fail	time	fail	time
5*	**0**	**0**	**0**	**0**	**0**	0.01	**0**	0.01
10*	1	**0.13**	1	0.14	1	0.16	**0**	0.23
15*	14	**1.44**	14	1.55	14	1.99	**0**	2.62
20*	712	**11.5**	546	13.79	747	24.33	**0**	18.49
25*	4	65.54	18	**36.88**	173	41.79	**0**	130.66

n	(d) Soft Latin square problem							
	AC* on $\mathcal{P}$		FDAC* on $\mathcal{P}$		EDAC* on $\mathcal{P}$		2-AC$_c^*$ on $\mathcal{P}^c$	
	fail	time	fail	time	fail	time	fail	time
3	5	**0**	**4**	**0**	**4**	**0**	5	**0**
4	172	**0.01**	100	0.02	**98**	0.03	125	0.02
5	25016	**2.61**	10038	2.85	**7325**	5.08	14950	2.78
6	751412	153.82	111545	105.95	**76353**	144.6	203839	**93.93**
7	-	-	1.87M	3558.49	**950480**	3632.47	3.62M	**2933.09**

Table 1(a) shows the results on the (m, n) instances of Langford's problem solved using a classical CSP model [8] containing $m \times n$ variables. In the problem, only a few instances are satisfiable (marked with "*"). Therefore, we soften the problem as described in Section 4. For each soft instance, we generate 10 instances and report the average results in Table 1(b). For both classical and soft cases, 2-AC$_c^*$ achieves the fewest number of fails among all four local consistencies in most instances. This shows that it does more prunings than AC*, FDAC*, and EDAC*, reduces more search space, and transmits both pruning and cost projection information better. The 2-AC$_c^*$ is clearly the most efficient for the larger and more difficult instances, which require more search efforts to either prove unsatisfiability or find an optimal solution. We improve the number of fails and runtime of, say, $(3, 14)$, by factors of 7.2 and 2.2 respectively. There are even instances that enforcing AC*, FDAC*, and EDAC* on $\mathcal{P}$ cannot be solved before timeout but enforcing 2-AC$_c^*$ on $\mathcal{P}^c$ can. The reduction rates of number of fails and runtime of $\mathcal{P}^c$ to $\mathcal{P}$ increase with the problem size. Exceptions are the "*" instances, in which once an $\perp$ cost solution is found, we need not prove its optimality and can terminate immediately. Such instances require relatively fewer search efforts, and the overhead of an extra model may not be counteracted.

Table 1(c) and 1(d) show the experimental results of classical and soft Latin square problem (prob003 in CSPLib) respectively. Contrary to Langford's problem, Latin square problem has many solutions. We assert preferences among the solutions by assigning to each allowed binary tuple a random cost from $\perp$ to n inclusive. We can see from Table 1(c) that classical Latin square problem is easy to solve up to $n = 25$. The amount of search is small even using a single model. Enforcing 2-AC$_c^*$ on $\mathcal{P}^c$ can still reduce the search space to achieve no backtracks. However, the runtime is not the fastest due to the overhead of an extra model. The soft Latin square problem is more difficult than the classical one since we are searching for the most preferred solution. We can solve only the smaller instances within 2 hours. Although 2-AC^*_c does not achieve the smallest number of fails among other local consistencies, its runtime is highly competitive as shown in Table 1(d). Like in the case of the classical and soft Langford's problem, 2-AC^*_c is the most efficient for the larger instances. In fact, even with two models, the time complexity of 2-AC^*_c is only a constant order higher than AC*, while FDAC* and EDAC* have higher time complexities $O(n^3d^3)$ and $O(n^2d^2 \max\{nd, \top\})$ respectively. In this problem, 2-AC^*_c strikes a balance between the amount of prunings and the time spent on discovering these prunings. This explains why 2-AC^*_c has more fails than EDAC* but the fastest runtime.

6 Conclusion

While we can rely on the standard propagation algorithms of the channeling constraints for classical CSPs to transmit pruning information between submodels, we have shown that this approach does not work well when combining WCSPs. Instead, we have generalized NC* and AC* and proposed the strictly stronger m-NC^*_c and m-AC^*_c respectively for combined models containing m submodels. 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 for *hard* instances.

Redundant modeling for WCSPs is a new concept and has plenty of scope for future work. We can investigate how generalized model induction can generate induced models of non-PermWCSPs, and to apply m-AC^*_c to the resultant combined models. It would be also interesting to incorporate c-supports and c-projections to FDAC* and EDAC* to obtain m-$FDAC^*_c$ and m-$EDAC^*_c$.

References

1. Cheng, B., Choi, K., Lee, J., Wu, J.: Increasing constraint propagation by redundant modeling: an experience report. Constraints **4**(2) (1999) 167–192
2. Schiex, T.: Arc consistency for soft constraints. In: Proc. of CP'00. (2000) 411–424
3. Larrosa, J.: Node and arc consistency in weighted CSP. In: Proc. of AAAI'02. (2002) 48–53
4. Law, Y., Lee, J.: Model induction: a new source of CSP model redundancy. In: Proc. of AAAI'02. (2002) 54–60
5. Larrosa, J., Schiex, T.: In the quest of the best form of local consistency for weighted CSP. In: Proc. of IJCAI'03. (2003) 239–244
6. de Givry, S., Heras, F., Zytnicki, M., Larrosa, J.: Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. In: Proc. of IJCAI'05. (2005) 84–89
7. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: hard and easy problems. In: Proc. of IJCAI'95. (1995) 631–637
8. Hnich, B., Smith, B., Walsh, T.: Dual modelling of permutation and injection problems. JAIR **21** (2004) 357–391