, , 1–27 ()
© Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Increasing Constraint Propagation by Redundant Modeling: an Experience Report

B. M. W. CHENG, K. M. F. CHOI, J. H. M. LEE AND J. C. K. WU

{mfchoi,jlee}@cse.cuhk.edu.hk

Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong, China

Editor:

Abstract. This paper describes our experience with a simple modeling and programming approach for increasing the amount of constraint propagation in the constraint solving process. The idea, although similar to redundant constraints, is based on the concept of redundant modeling. We introduce the notions of CSP model and model redundancy, and show how mutually redundant models can be combined and connected using channeling constraints. The combined model contains the mutually redundant models as sub-models. Channeling constraints allow the sub-models to cooperate during constraint solving by propagating constraints freely amongst the sub-models. This extra level of pruning and propagation activities becomes the source of execution speedup. We perform two case studies to evaluate the effectiveness and efficiency of our method. The first case study is based on the simple and well-known n-queens problem, while the second case study applies our method in the design and construction of a real-life nurse rostering system. Experimental results provide empirical evidence in line with our prediction.

Keywords: Constraint propagation, redundant modeling

1. Introduction

The task at hand is to tackle *constraint satisfaction problems* (CSP) defined in the sense of Mackworth [42], which can be stated briefly as follows:

> *We are given a set of variables, a domain of possible values for each variable, and a conjunction of constraints. Each constraint is a relation defined over a subset of the variables, limiting the combination of values that the variables in this subset can take. The goal is to find a* consistent *assignment of values to the variables so that all the constraints are satisfied simultaneously.*

CSPs are, in general, NP-hard [13]. Thus, a general algorithm designed to solve any CSP will likely require exponential time in problem size in the worst case. One widely-adopted approach in solving CSPs features various degrees of combinations of backtracking tree search and constraint propagation [47, 37]. This framework is realized in the constraint logic programming language CHIP [18] and C++ class library ILOG SOLVER [31], which have been applied successfully to real-life industrial applications [16, 17, 48, 39, 2, 38].

There are at least six approaches to increase the efficiency of constraint solving. The first approach amounts to incorporating variable and value ordering heuristics [6, 30] in the labeling procedure of a propagation-based constraint solver. In-

corporation of these heuristics requires extra programming effort and these heuristics are usually problem and domain specific. The second approach is to introduce redundant constraints whenever possible. This technique has not so far been formally described or studied but one can find it used in many places in the literature, such as [52, 3]. While this approach could sometimes exhibit surprisingly good results, a problem is that there is no systematic way of generating useful redundant constraints for a general CSP. The third approach exploits such problem specific characteristics as structure of the constraint graph [14, 21] and properties of the constraints and domains [53, 43, 36]. Other problem characteristics include decomposability [51], symmetries and interchangeability [4, 22, 20]. This class of techniques is also problem and domain specific. The fourth approach enhances basic backtracking search with such techniques as backmarking [23, 24, 30] and backjumping [25, 26, 49]. The fifth approach is to entirely replace propagation-based techniques, which are algorithmic in nature, by the promising stochastic techniques such as the iterative repair method [45, 41]. The sixth approach employs cooperation among different constraint solvers [44, 40]. The last three approaches represent partial or complete modification to the underlying constraint solving mechanism. This type of technological innovation is not easy to come up with.

Propagation-based constraint languages and tools are widely available. The aim of this research project is to devise an approach for speeding up solution search in CSPs while adopting existing technology and tools. We observe that there are usually more than one way of modeling a problem as a CSP. By *modeling* a problem as a CSP, we mean the process of determining the variables, the associated domains of the variables, and the expressions of the constraints. Experiences suggest that some constraints can be expressed succinctly in one model and yet the same constraints result in a cumbersome specification in another model.

Our approach is inspired by the effectiveness of redundant constraints[1] in increasing constraint propagation since they augment the pruning of the original constraints of a CSP. Intuitively speaking, our method amounts to implementing more than one model of a given problem and connect the model implementations. Theoretically, not pragmatically, any one model implementation suffices to solve the CSP at hand completely. Therefore, the models are redundant with respect to one another. Each model implementation corresponds to a constraint network. In order for the different models to cooperate during constraint solving, the network must be connected so that pruning can be propagated among the networks in a multi-directional manner. One way to achieve a multi-directional connection is to use constraints to express the relationship among variables in different models. We call such constraints, which are not part of the original problem specification and exist only as an artifact of network connection, *channeling constraints*. To confirm the effectiveness and demonstrate the feasibility of our proposal, we present two case studies of our method. The first case study is based on the simple and well-known n-queens problem, while the second case study applies our method in the implementation of a working nurse rostering system for the emergency ward of a local hospital. This paper is a revised and extended version of [11].

The rest of the paper is organized as follows. Section 2 reviews related techniques that use multiple viewpoints to improve problem-solving efficiency. Section 3 presents the notions of CSP model and model redundancy. Channeling constraints are introduced. We also discuss issues pertaining to the use of redundant modeling. In section 4, we present the first case study and its benchmark results, which confirm increased pruning with our method. Using the 9-queens problem, we animate the propagation behavior of redundant modeling in section 5. Section 6 presents the second case study. We first give the domain description of the nurse rostering problem, followed by a description of two single models and one combined model. We end the section by discussing our implementation experience of the three models and some benchmarking results. Section 7 summarizes our contributions and sheds light on future direction of research.

2. Related Work

Geelen [27] introduces two improved value and variable ordering heuristics for solving CSPs. He also introduces a dual-viewpoint approach for a special class of binary CSPs, namely permutation problems. This approach allows suitable extension to many heuristics, including the two presented in the earlier part of the paper. A binary CSP is a *permutation problem* if the domains of all variables are the same, the number of variables is equal to the size of the domain, and each variable should take on a different value. In other words, every value must be assigned to exactly one variable. We can consider a permutation problem from two different perspectives. In the normal viewpoint, we solve the problem by finding a value for every variable. In the inverted viewpoint, we solve the problem by finding a variable for each value. Geelen improves the most-constrained-first variable ordering heuristic by calculating and incorporating the constrainedness information from both viewpoints; hence reducing the number of backtracking.

Eaton and Freuder [19] consider a team of cooperative constraint-based reasoning agents, each of which has a different representation of a CSP. They explain the advantages of having diverse representations members of a team, in terms of fault-tolerance, efficiency, and flexibility. Diverse representations can also compensate for incomplete knowledge by sharing information obtained during problem-solving. Eaton and Freuder use logic puzzle problems to demonstrate how a team of such agents can cooperate to find solutions to a CSP even when each member of the team has incomplete knowledge of the CSP.

Caseau and Koppstein [9] propose and implement three models[2] to tackle the Time-Constrained Traveling Salesman Problem (TCTSP). Each implementation works well on only some instances of TCTSP. Caseau and Koppstein improve the robustness by "combining" the models. Out of the three models, one is the base model. Their combination is made possible by the LAURE language [7, 8] which allows the binding of constraints and algorithms with rules. Their work differs from our proposal in three respects. First, Caseau and Koppstein study the combination of only specific models, whereas we propose channeling constraints as a vehicle for connecting general models. Second, sub-models in their combined model do not

share equal partnership since the base model plays a more dominant role. Third, their model combination technique relies on the feature of a particular programming language, whereas our method should work in any constraint-based language.

Although our method does not rely on any particular programming language feature, appropriate language constructs can help expressing channeling constraints more succinctly. For example, the `element` constraint [31] (its usage discussed in section 4.2) and the `IlcInverse` function [33] provided in the ILOG Solver library serve this purpose. In particular, the latter, available only in version 3.2 or above, allows concise expression of channeling constraints relating models that are inverse of each other. Unfortunately, the `IlcInverse` function was not yet available during our construction of the nurse rostering system.

Last but not least, Jourdan's [34, 35] work on multiple modeling is most similar to the philosophy of our approach. Programs representing different but redundant views of the same problem are synchronized using the communication mechanisms of constraint logic programming (CLP) and concurrent constraint (CC) languages. Besides redundant models, Jourdan also explores the notions of cooperating and hierarchical models.

3. Redundant Modeling

Let P be a problem. By modeling P as a CSP, we need to determine the set of variables $\mathcal{X}$, a function $\mathcal{F}_{\mathcal{X}}$ that associates a domain D for each variable $X \in \mathcal{X}$, and the set of constraints $\mathcal{C}_{\mathcal{X}}$ expressed using variables in $\mathcal{X}$. The triple $\langle \mathcal{X}, \mathcal{F}_{\mathcal{X}}, \mathcal{C}_{\mathcal{X}} \rangle$ is called a *model* M of P. An important requirement of a model M of problem P is that M must provide a complete characterization of the solution space of P. There are usually more than one possible model of P. Take P as a generic job-shop scheduling problem. We can either model P as assigning machines to jobs or assigning jobs to machines. In the first model, we have jobs as variables, taking on machines as values. In the second model, we have machines as variables, taking on jobs as values[3]. Two models $M_1 = \langle \mathcal{X}_1, \mathcal{F}_{\mathcal{X}_1}, \mathcal{C}_{\mathcal{X}_1} \rangle$ and $M_2 = \langle \mathcal{X}_2, \mathcal{F}_{\mathcal{X}_2}, \mathcal{C}_{\mathcal{X}_2} \rangle$ of P are *redundant* if M_1 and M_2 are models of P respectively. A *channeling constraint* c for $M_1 = \langle \mathcal{X}_1, \mathcal{F}_{\mathcal{X}_1}, \mathcal{C}_{\mathcal{X}_1} \rangle$ and $M_2 = \langle \mathcal{X}_2, \mathcal{F}_{\mathcal{X}_2}, \mathcal{C}_{\mathcal{X}_2} \rangle$ is a constraint relating variables of $\mathcal{X}_1$ and $\mathcal{X}_2$, where $c \notin \mathcal{C}_{\mathcal{X}_1}$ and $c \notin \mathcal{C}_{\mathcal{X}_2}$. Consider a problem P and a set $\{M_1, \ldots, M_n\}$ $(n > 1)$ of mutually redundant models of P. A *combined model* $M_{\{M_1,\ldots,M_n\}}$ of P is obtained by implementing the set of models $\{M_1, \ldots, M_n\}$ $(n > 1)$ of P and connecting these models with channeling constraints. Note that additional propagation may be resulted only when different models are connected. By combining two models which are essentially the same (modulo variable renaming), one does not gain extra pruning.

In a standard implementation of constraint propagation augmented with variable labeling, constraints are propagated once they are posted to the constraint store. When no further propagation is possible, an unbound variable x is selected. A value is picked from the domain D_x of x and instantiated to x. This new binding is in turn propagated to the rest of the constraint store. If any domain becomes empty, then the last instantiation of x and all its consequences must be undone,

and another value in D_x is tried. If there is no more value in D_x, then we backtrack to the next value of the previously selected variable. Note that if the domain of any variable y becomes a singleton $\{v\}$ at any stage during propagation, then y is instantiated to v automatically. Variable and value ordering heuristics determine which variable and value to pick next.

Node- and arc-consistency techniques [55, 42, 43, 46, 15, 5] enforce the consistency of nodes (variable domains) and arcs (constraints) in a constraint network by reducing variable domains. Constraint propagation is essentially an iterative algorithm that propagate the domain reduction information throughout the constraint network. An important property of constraint propagation is that the order in which constraints are propagated is unimportant [42]. In other words, the final state of a constraint network is independent of the order of constraint propagation. This property allows us to understand the propagation behavior of a combined model with two sub-models, model one and two, as follows.

1. Propagate constraints in model one until no more propagation is possible.
2. Propagate the variable instantiation and domain reduction information in model one, if any, to model two via either the shared variables in the models or the channeling constraints.
3. Propagate constraints in model two until no more propagation is possible.
4. Propagate the variable instantiation and domain reduction information in model two, if any, to model one via either the shared variables in the models or the channeling constraints.
5. Repeat steps 1 to 4 until no more propagation is possible.
6. Label the next variable, possibly using some form of variable/value ordering heuristics, and goto step 1.

The above procedure can be generalized for a combined model with more than two sub-models in a straightforward manner.

In a combined model, we differentiate between two types of propagation: intra-model propagation and inter-model propagation. *Intra-model* propagation propagates variable instantiation and domain reduction information to other variables within the same sub-model; while *inter-model* propagation transmits the pruning information from one model to another via either the shared variables of the models or the channeling constraints. In the above algorithm, steps 1 and 3 contain intra-model propagation while steps 2 and 4 perform inter-model propagation.

In the following, we share our experience in using the redundant modeling method and raise awareness of issues that need to be considered when adopting the method. We also elaborate on the relationship between redundant models and redundant constraints.

3.1. How Many Models?

A first question is how many redundant models should be used. Consider two models M_1 and M_2 of a problem P. Since M_1 and M_2 are complete specification

of P, an implementation of the combined model $M_{\{M_1,M_2\}}$ is guaranteed to be at least as efficient as an implementation of either model alone in principle since the individual models are embedded in the combined model. Therefore, it seems reasonable to combine and implement as many mutually redundant models as one can dream of. One must, however, take into account the time and memory overhead of larger network size in the combined model. In addition, implementing an alternate model is not a task that can be taken lightly. However, in our experience, the time spent on identifying the features of a practical application exceeds the time spent on designing and implementing the alternate model by an order of magnitude.

3.2. How to Come Up with Redundant Models?

Another common and important question to ask is how to generate a redundant model. We observe that many real-life scheduling and resource allocation problems can be modeled reciprocally in the sense that if a problem can be modeled as assigning objects of type X to objects of type Y, then the same problem can also be modeled as assigning objects of type Y to those of type X. This principle is general and widely applicable. At the beginning of this section, we have already given an example of applying the principle to a generic job-shop scheduling problem. Similarly, we can model a train driver rostering problem as either assigning trips to drivers or assigning drivers to trips. We can also model a school timetabling problem as either assigning courses to time slots in the timetable or assigning a set of time slots to courses. In the channel assignment problem, a type of configuration problems, we can assign channels to nodes or nodes to channels. We can find more examples of this type. It may be argued that the above resembles the resource-centered view and the job-centered view used in the scheduling community. The novelty of our proposal is, however, not on suggesting multiple perspectives on the same problem since others had proposed similar ideas. The novelty is on suggesting that these perspectives can be connected to form a combined perspective which exhibits more efficient pruning behavior.

Using reciprocal views of problems is one way of obtaining redundant models but it is not the only way. The general guideline is to study the problem at hand from different angles and perspectives. For example, Tsang [50] suggests two ways of modeling the 8-queens problem. The first and familiar model consists of eight variables, each of which denotes the column position of a queen in a different row of the chess board. Every domain contains the eight possible positions of a queen in a row. An alternative model consists also of eight variables, each of which denotes the position of the queen on either one of the sixty-four squares on the chess board. Thus, the domain of each variable becomes $\{1, \ldots, 64\}$. It is not difficult to come up with yet another model consisting of sixty-four variables, each of which has domain $\{0, 1\}$. Each variable denotes a square on the board. A value of 0 denotes an empty square and a value of 1 denotes a square occupied by a queen.

To help illustrate our points, we present four different models for a real-life application, the *Mystery Shopper Problem*, as follows. A well-known cosmetic company wants to evaluate the performance of her saleladies, who station at the company's

counters at various department stores in different geographical locations. For this purpose, the company has hired some secret agents to disguise as shoppers to visit the salesladies. Each shopper has to visit more than one saleslady and each saleslady has to be visited by more than one shopper. The visits must be scheduled in such a way that each saleslady must be visited by shoppers of different varieties and that the visits should be spaced out roughly evenly. Also, shoppers should visit salesladies in different geographic locations. In the following, we only specify the variables and the associated domains of each model. The constraints can be coded accordingly.

- Model 1: each variable `VisitSaleslady` denotes the shopper visiting a particular saleslady on a particular visit; and the domain of each variable is the set of all shoppers.
- Model 2: each variable `ShopperVisit` denotes the saleslady to be visited by a particular shopper on a particular visit; and the domain of each variable is the set of all saleslady. Model 2 is, in a sense, a reciprocal view of model 1.
- Model 3: each variable `ShopperSaleslady` denotes whether a particular shopper visits a particular saleslady, and, if so, the visit number of the shopper; and the domain of each variable is the set $\{0, 1, \ldots, m\}$, where m is the number of saleslady visits required of each shopper and a visit number of 0 implies that the shopper does not visit the saleslady.
- Model 4: each variable `SalesladyShopper` denotes whether a particular saleslady is visited by a particular shopper, and, if so, the visit number of the salelady; and the domain of each variable is the set $\{0, 1, \ldots, n\}$, where n is the number of secret shopper visits imposed on each saleslady and a visit number of 0 implies that the saleslady is not visited by the shopper. Model 4 is a reciprocal view of model 3. Note that n is not necessarily equal to m as specified in model 3.

3.3. How to Come Up with Channeling Constraints?

Channeling constraints for models M_1 and M_2 must be able to propagate constraints from M_1 to M_2 and vice versa. Effective channeling constraints depend on the structure of the respective models. Suppose M_1 is modeled after assigning objects of type X to those of type Y and M_2 is modeled after assigning objects of type Y to those of type X. Effective channeling constraints for these two particular redundant models are of the form:

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

This form of constraint is simple and can be generated systematically by enumerating the possible objects of type X and objects of type Y. This amounts to a simple single or double loop using most programming languages.

As further examples, we list in the following how the models in the Mystery Shopper Problem can be connected with each other. For each pair of models, the general

form of the channeling constraints is given. Our variable naming convention is that `Shopper` and `Saleslady` are actual names of a shopper and a saleslady, and `Visit` is a visit number. Therefore, `VisitSaleslady` denotes all possible combination of salesladies and visit numbers. An actual variable might be named `threeClara`, which denotes the third visit to the saleslady Clara.

- Models 1 and 2:

$$\texttt{VisitSaleslady} == \texttt{Shopper} \Longleftrightarrow \texttt{ShopperVisit} == \texttt{Saleslady}.$$

- Models 1 and 3:

$$\texttt{VisitSaleslady} == \texttt{Shopper} \Longleftrightarrow \texttt{ShopperSaleslady} == \texttt{Visit}.$$

- Models 1 and 4:

$$\texttt{VisitSaleslady} == \texttt{Shopper} \Longleftrightarrow \texttt{SalesladyShopper} == \texttt{Visit}.$$

- Models 2 and 3:

$$\texttt{ShopperVisit} == \texttt{Saleslady} \Longleftrightarrow \texttt{ShopperSaleslady} == \texttt{Visit}.$$

- Models 2 and 4:

$$\texttt{ShopperVisit} == \texttt{Saleslady} \Longleftrightarrow \texttt{SalesladyShopper} == \texttt{Visit}.$$

- Models 3 and 4:

$$\texttt{SalesladyShopper} == \texttt{Visit} \Longleftrightarrow \texttt{SalesladyShopper} == \texttt{Visit}.$$

3.4. Relationship with Redundant Constraints?

Note that redundant constraints and redundant models are orthogonal concepts in general although their working principles are similar. Consider two redundant models $M_1 = \langle \mathcal{X}_1, \mathcal{F}_{\mathcal{X}_1}, \mathcal{C}_{\mathcal{X}_1} \rangle$ and $M_2 = \langle \mathcal{X}_2, \mathcal{F}_{\mathcal{X}_2}, \mathcal{C}_{\mathcal{X}_2} \rangle$ of a problem P. Constraints in $\mathcal{C}_{\mathcal{X}_1}$ and those in $\mathcal{C}_{\mathcal{X}_2}$ have no relationship with each other in general since $M_1 \neq M_2$. Within a model, say M_1 (or M_2), we can add a redundant constraint c such that c is entailed by $\mathcal{C}_{\mathcal{X}_1}$ (or $\mathcal{C}_{\mathcal{X}_2}$).

Having made this claim, we must point out that, under certain formulation of channeling constraints (to be described in our case studies), $\mathcal{C}_{\mathcal{X}_1}$ (or $\mathcal{C}_{\mathcal{X}_2}$) could be regarded as redundant constraints of $\mathcal{R}$ plus $\mathcal{C}_{\mathcal{X}_2}$ (or $\mathcal{C}_{\mathcal{X}_1}$), where $\mathcal{R}$ is the set of channeling constraints for M_1 and M_2. Note that this point of view may explain the efficiency of the redundant modeling approach. The assertion, however, is unintuitive and unnatural since $\mathcal{R}$ is not part of the problem specification and is created only artificially for model connection purpose.

4. Case Study 1

In the first case study, we consider the familiar n-queens problem, the task of which is to place n queens on an $n \times n$ chess board so that no two queens can attack each other. In other words, no pair of queens can be on the same row, the same column or the same diagonal (positive or negative). The choice of this simple and well-known problem allows us to study redundant modeling systematically. We construct four implementations, one based on a single model and three based on a combined model with different variable labeling heuristics. We verify the improved pruning behavior offered by redundant modeling both in terms of the number of failures and choice points. However, two implementations using redundant models execute slower than the implementation based on a single model. For this particular problem, the gain obtained from less choice points and failures is counteracted by the overhead resulted from more variables and constraints in the combined model. This case study also demonstrates a distinguished feature of redundant modeling: we are able to define a new type of value ordering heuristic made possible only by redundant modeling. The implementation based on this new heuristic exhibits the best time performance among the four implementations.

4.1. Modeling

In this section, we describe two different but symmetrical models of the n-queens problem. Then, a combined model is obtained by connecting these two models with channeling constraints.

4.1.1. Model One This model consists of n variables x_i $(i = 0, 1, \ldots, n-1)$, each of which denotes the column position of a queen in a different row of the chess board. The domain D_{x_i} of variable x_i contains the n possible positions of queen i in row i. In this formulation, the rule that no two queens are on the same row is implicitly satisfied. The other rules are expressed as the following constraints:

1. $x_i \neq x_j$ for all $i \neq j$ and $i, j = 0, 1, \ldots, n-1$.
2. $|x_i - x_j| \neq |i - j|$ for all $i \neq j$ and $i, j = 0, 1, \ldots, n-1$.

Rules 1 and 2 specify that no pair of queens can be in the same column and in the same diagonal (positive or negative) respectively.

4.1.2. Model Two Model two is just the reciprocal view of model one. Again, there are totally n variables, namely $y_0, y_1, \ldots, y_{n-1}$. Each variable denotes the row position of each queen in a different column. The domain D_{y_i} of variable y_i contains the n possible positions of queen i in column i. Similar to model one, the rule that no pair of queen is on the same column is implicitly satisfied. The other constraints are expressed as follow:

1. $y_i \neq y_j$ for all $i \neq j$ and $i, j = 0, 1, \ldots, n-1$.

2. $|y_i - y_j| \neq |i - j|$ for all $i \neq j$ and $i, j = 0, 1, \ldots, n-1$.

We can easily check that model two is a symmetrical variant of model one.

4.1.3. A Combined Model A combined model can be obtained by connecting models one and two with the following channeling constraints:

$$x_i == j \iff y_j == i$$

for $i, j = 0, 1, \ldots, n-1$. This set of channeling constraints are used to relate variables in the two models so that constraints can propagate freely between the models.

Note that each variable in model one has a corresponding value in model two and vice versa. Besides, the channeling constraints connect every variable in model one with every variable in model two for every possible combination of values. Under this formulation, constraints in model one (or two) can be regarded as redundant constraints of the channeling constraints plus constraints in model two (or one). Such a relationship does not necessarily hold true in general as programmers may choose to connect only certain subsets of variables or values.

4.2. Implementation and Benchmarking Results

The various models are realized in C++ with ILOG Solver library 3.0 [31]. A naive implementation would result in n^2 channeling constraints. We reduce the number of channeling constraints to $2n$ by utilizing the `element` constraint [31] which maps the value of one variable to a function of another variable. Therefore the size of the combined model is two times the size of the single model plus $2n$.

Combining different models with different variable labeling heuristics, we come up with four implementations. The first implementation is based on model one, while the second, third and fourth implementations are based on the combined model. Model two is not implemented since it is identical to model one. Implementations one to three employ the simple smallest-domain-first variable ordering heuristic and the smallest-first value ordering heuristic in the variable labeling process. Since the combined model contains models one and two as sub-models, it suffices to label only variables in either models. We label only variables in model one in the second implementation. In the third implementation, we consider variables from both models in the labeling step.

The fourth implementation also employs the smallest-domain-first variable ordering heuristic but we use a new dynamic value ordering heuristic by Jourdan [35]. The new heuristic makes use of the fact that every value in the domain of a variable corresponds to a variable in the alternative model. For example, the value j in D_{x_i} in model one corresponds to the variable y_j in model two. The idea of the heuristic is to order values in a variable domain using the domain sizes of the corresponding variables in the alternative model. When we have to select a value from the domain of a variable, we choose the value, whose corresponding variable in the alternative

Table 1. Number of Failures.

n	Single Model	Combined Model		
	1st Impl.	2nd Impl.	3rd Impl.	4th Impl.
5	0	0	0	0
10	9	7	6	6
15	4	3	3	13
20	33	20	20	4
25	62	47	8	7
30	29	16	66	0
35	68	26	35	26
40	19	10	254	14
45	76	6	36	3
50	512	130	2	2
55	213	5	37	10
60	406	143	4918	0
65	1	1	2028	11
70	1	1	-	1
75	2219	1443	-	0
80	21	7	-	10
85	6	1	-	8
90	341	125	-	20
95	2288	0	-	18
100	22	9	-	69
105	45829	34026	-	5
110	-	-	-	2
115	-	-	-	2781
120	-	-	-	0

model has the smallest domain. In the fourth implementation, we label only variables in model one and order values in a domain using the dynamic domain sizes of variables in model two.

The number of failures, the number of choice points and the CPU time (in seconds) of various implementations are shown in tables 1, 2 and 3 respectively. The first column of each table indicates the size of the problem instances. The remaining four columns contain the relevant data of each table for each of the four implementations. Entries marked with "-" correspond to problem instances that cannot be solved within an hour. The benchmarking results confirm that the combined model, except for some cases in the third implementation, always give us less number of failures and choice points. In other words, redundant modeling increases the amount of constraint propagation. The problem with the third implementation is that variables from both models are considered for instantiation, resulting from many more variables with the same domain size. This may reduce the accuracy and effectiveness of the smallest-domain-first variable ordering heuristic. We are currently investigating the exact cause for the breakdown of the heuristic.

However, the CPU time for the first two implementations of the combined model is longer than that of the single model. This is due to the fact that there are many

Table 2. Number of Choice Points.

n	Single Model	Combined Model		
	1st Impl.	2nd Impl.	3rd Impl.	4th Impl.
5	2	2	2	2
10	15	13	11	11
15	12	10	10	21
20	43	30	31	15
25	79	64	25	24
30	51	38	86	24
35	95	51	59	54
40	50	41	280	47
45	113	39	68	38
50	553	169	37	44
55	256	51	79	55
60	456	193	4960	52
65	59	59	2079	68
70	61	61	-	62
75	2278	1502	-	65
80	89	75	-	82
85	83	74	-	84
90	416	199	-	99
95	2375	86	-	104
100	115	100	-	161
105	45918	34114	-	98
110	-	-	-	103
115	-	-	-	2882
120	-	-	-	116

more constraints and variables in the combined model. Thus more time is needed to set up and maintain a larger constraint network during constraint propagation. Unfortunately, this overhead surpasses the extra pruning gained from redundant modeling. Besides, there is little searching needed for the n-queens problem when n is less than 130. That is why the effect of extra pruning is not outstanding.

The fourth implementation is the most efficient among the four implementations. It is able to solve some problem instances which cannot be solved by the other implementations. Redundant modeling, not only increase pruning but also, enables us to define a better value ordering heuristic. This verifies that redundant modeling can be integrated with other constraint solving techniques seamlessly and benefits from each other. We are aware of other variable labeling heuristics for the n-queens problem. For example, [32] suggests a variant of the smallest-domain-first variable ordering heuristic: choose the variable with the smallest minimum value among those that have the smallest domain size. Aggoun [1] suggests using a middle-out value ordering heuristic, in which values are ordered ascendingly and middle values are picked before smaller and larger values. These heuristics are difficult to understand. It is unclear why they work well with the n-queens problem; whereas

Table 3. CPU Time (in seconds).

n	Single Model	Combined Model		
	1st Impl.	2nd Impl.	3rd Impl.	4th Impl.
5	0	0.01	0	0
10	0	0.01	0.01	0.01
15	0	0.02	0.02	0.03
20	0.02	0.06	0.06	0.03
25	0.03	0.13	0.07	0.06
30	0.02	0.11	0.22	0.08
35	0.03	0.16	0.2	0.17
40	0.03	0.2	0.86	0.19
45	0.05	0.26	0.33	0.24
50	0.23	0.67	0.32	0.32
55	0.12	0.43	0.53	0.44
60	0.23	0.97	17.54	0.53
65	0.04	0.66	8.46	0.67
70	0.07	0.81	-	0.79
75	1.25	6.68	-	0.96
80	0.08	1.19	-	1.17
85	0.08	1.42	-	1.4
90	0.31	2.5	-	1.75
95	1.21	1.94	-	2.02
100	0.13	2.31	-	2.55
105	31.37	240.83	-	2.64
110	-	-	-	3.1
115	-	-	-	26.3
120	-	-	-	3.88

the new heuristic based on redundant modeling represents a simple application of the first-fail principle [30].

5. Illustrating Inter-Model Propagation

The simplicity of the n-queens problem makes it suitable for illustrating the propagation behavior of redundant modeling. Before we move on to the second case study, we use the 9-queens problem to explain how redundant modeling achieves additional pruning in constraint propagation.

To depict the pruning behavior offered by redundant modeling in more details, let us consider the 9-queens problem and compare its execution in the single model (first implementation) and combined model (second implementation). We denote variables in model one by x_i and variables in model two by y_i, where $i = 0, 1, \ldots, 8$. The domain of each variable is $\{0, 1, \ldots, 8\}$. In the following, we use a series of diagrams to illustrate the states of variable domains at various stages during execution. Each domain value corresponds to a square in the diagrams. The different possible states of domain values are represented using different shadings as follows:

▣ = assigned value

⊠ = tried value

■ = value eliminated by instantiation in model one

■ = value eliminated by instantiation in model two

■ = value eliminated by constraint propagation in model one

■ = value eliminated by constraint propagation in model two

□ = untried value

Having two shadings in a square means that the explanation from both shadings is applicable to that square.

After all constraints are posted to the constraint store, no pruning occurs. Execution goes on to assign 0 to x_0, 2 to x_1 (since 0 and 1 are pruned by the instantiation of x_0), and 4 to x_2 (since 0, 1, 2, and 3 are pruned by the instantiation of x_0 and x_1). The next selected variable is x_5, which is instantiated with the value 3. Further execution confirms that this instantiation leads to failure. Figure 1 shows the states of the single and combined models after $x_5 = 3$ is rejected.

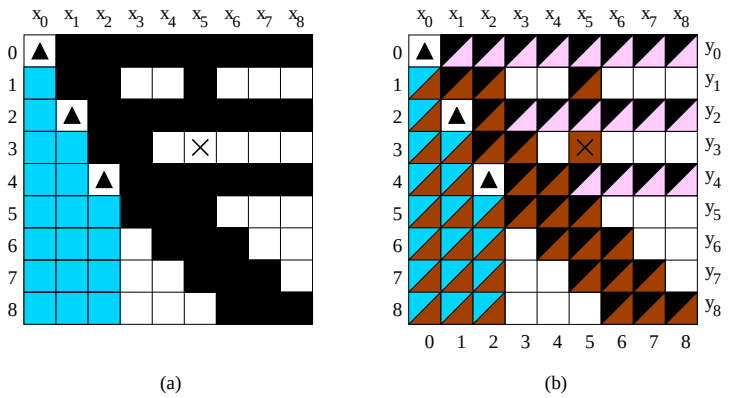

Figure 1. The states of (a) the single model and (b) the combined model after $x_5 = 3$ is rejected.

Since the domain of x_5 becomes the singleton $\{8\}$, x_5 is automatically bound to 8 and constraint propagation starts again. From this point onward, the behavior of the single model and the combined model becomes different. In the single model, constraint propagation stops at the state as shown in figure 2. Execution continues to label other variables until it discovers that the previous assignment $x_2 = 4$ is unfruitful.

In the combined model, constraints are propagated between both sub-models. In model one, intra-model propagation stops at exactly the same state as the single model as shown in figure 2. The propagation process does not end here, however. This state implies that the only available column position in row 6 is 8. This

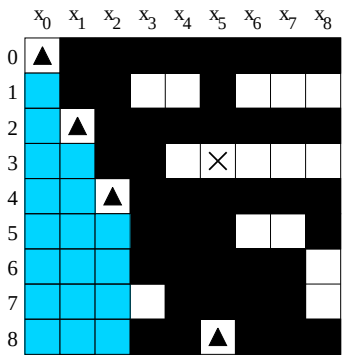

Figure 2. The state of the single model after x_5 is assigned with 8.

information is propagated via the channeling constraints, *i.e.* using inter-model propagation, to model two and the domain of y_6 becomes the singleton $\{8\}$, which induces further propagation in model two. Intra-model propagation takes place in model two until the state as shown in figure 3 is reached. Using inter-modeling

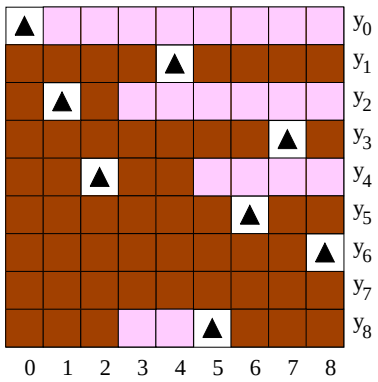

Figure 3. The state of model two after x_5 is assigned with 8.

propagation, this state information is transmitted back to model one via channeling constraints resulting in the state as shown in figure 4. Combining the two sub-states, we get the state of the combined model as shown in figure 5. In the states of sub-models or the combined model, the domains of both x_3 and y_7 are empty. Execution, instead of having to label other variables, can backtrack to try other values of x_2 immediately. Therefore, the extra pruning offered by redundant modeling helps us to detect failure earlier than the single model.

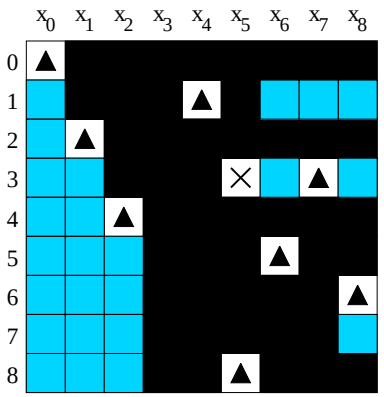

Figure 4. The state of model one after x_5 is assigned with 8.

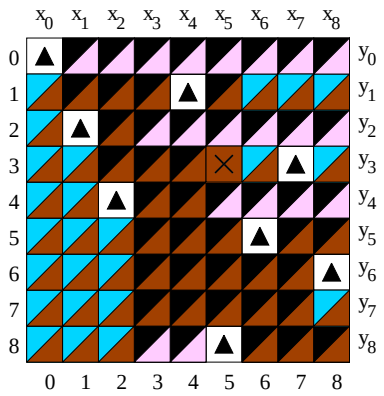

Figure 5. The state of the combined model after x_5 is assigned with 8.

6. Case Study 2

Our task is to design and implement a nursing staff rostering system for the Accident and Emergency Unit (AEU) of the Tang Shiu Kin Hospital (TSKH) in Hong Kong. The AEU provides daily 24-hour emergency services to the general public seven days a week. Therefore, AEU nurses have to work in shifts. The main function of the rostering system is to roster the nurses in such a way that steady and high-quality services are provided to the community, taking into account (1) professional rules, (2) preferential rules, (3) pre-arranged duties, and (4) pre-arranged preferred shifts. Other than having to obey all professional rules, fairness is an important measure

Table 4. A Sample Roster Duty Sheet

Nurse	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Nurse-01	E	A	O	P	A	N	PH
Nurse-02	VL	VL	VL	VL	VL	VL	O
Nurse-03	A	CO2	P	A	N	O	E
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

of the quality of the generated roster. Every nurse must be assured of equal chance in taking night shifts, having day-offs on weekends or the actual public holidays, *etc*, although most fairness rules are in the form of soft constraints. The main difficulties of the process arise from pre-arranged duties, vacation leaves, and pre-arranged preferred shifts requested by the nurses, which often break the regularities of wanted (or unwanted) duties. In the following, we give an overview of the nurse rostering system. Readers interested in how we handle soft constraints and fair rotation of wanted and unwanted shifts are referred to [10, 12].

6.1. Domain Description

There are three basic shifts in a day: namely AM shift (A), PM shift (P), and night shift (N). An evening shift (E) is essentially a PM shift with a slightly different duty time. An irregular shift (I) has special working hours either arranged by the nursing officer or requested by individual nurses. Other shift types concern holidays and special duties. Nurses can take several different types of holiday. These include day-off (O), compensation-off (CO), public holiday (PH) and vacation leave (VL). In addition, nurses can be pre-assigned some special work shifts such as study day (SD) and staff-on-loan (SOL). In total, there are eleven shift types.

A weekly duty roster for week i should be generated about two weeks prior to i. The rostering process assigns, for each nurse, a work shift for Monday to Sunday, taking into account of the nurse's past rostering history. A sample roster duty sheet is shown in table 4. There are two steps for a duty planner to complete a duty roster. First, the planner has to collect information about pre-assigned shifts and shift requests from the nurses. Second the planner generates the remaining shift slots, observing all planning rules and nurse preference rules.

There are two types of planning rules: imperative planning rules and preference planning rules. Imperative planning rules are rules that must be respected in any timetable. Therefore, in generating the roster, the duty planner must ensure that every planning decision made is coherent with these hard rules. There are totally six imperative rules. Some examples rules are:

Rule 1 *Each staff is required to work one shift per day.*

Rule 2 *Each staff gets one day-off per week.*

Preference planning rules are optional in the sense that they should be satisfied as much as possible. Violating preference rules, however, does not endanger the validity of a roster. Therefore, preference rules corresponds to soft constraints. A soft constraint c can be viewed as a disjunctive constraint $c \cup \texttt{true}$. Following Van Hentenrcyk [52], we model disjunctive constraints as choices. While choice is easy to implement in a constraint programming language, accumulation of soft constraints can result in a huge search tree. While the preference rules are not professional rules, they do reflect the preference of nurses in general. Therefore, the number of preference rules satisfied is a good measure of the quality of a generated roster. There are totally ten preference rules[4]. Some examples rules are:

Rule 3 *A nurse should not work for the same shift for two consecutive days. In particular, a nurse prefer alternating A shift or P/E shift. If such an arrangement is impossible, a nurse also accepts two consecutive A or P shifts. However, under no circumstance will three consecutive A or P/E shifts be accepted by a nurse.*

Rule 4 *Apart from all holiday schedule, the number of A shifts and P shifts allocated to each nurse each week should be balanced.*

6.2. Modeling

In this section, we describe two models for the nurse rostering problem. The first model is based on the format of the roster sheet, which suggests allocation of shifts to nurses. Implementation of the first model performs well in general but fails to return an answer in a timely manner (within 15 minutes) for some difficult cases. This prompts our work on the second model, which is an allocation of nurses to shifts. Implementation of the second model again exhibits deficiency on some problem instances while performing well in general. Having two mutually redundant models in hand, we proceed to connect the models using channeling constraints.

6.2.1. Model One In a roster sheet, each row consists of seven slots, holding the work shifts assigned to a nurse in a scheduled week. Each nurse occupies a row in the roster sheet. It is thus natural to model the slots *Nurse-DayOfWeek* on the sheet as constrained variables, each of which is associated with a domain of eleven possible shift types {A,P,N,E,I,O,CO,PH,VL,SD,SOL}. If there are n nurses in the AEU, then there will be $7n$ variables in the rostering system.

Under this formulation, we have rule 1 satisfied for free since the rule is implicitly satisfied by requiring each constrained variable to take on exactly one value. Rule 2 can be modeled using a counting constraint [31] as follows:

For each nurse `Nurse`*, the number of variables in the set*

$$\{\texttt{Nurse-Mon}, \texttt{Nurse-Tue}, \ldots, \texttt{Nurse-Sun}\}$$

assigned with the day-off (O) shift must be equal to one.

Preference planning rules, or soft constraints, are expressed in the same manner as imperative planning rules. There is, however, one significant difference in how they are posted to the constraint solving engine. For each soft constraint c, we set up a choice point. In the first branch, the constraint c is told to the solver. The other branch, one without c, is tried upon backtracking.

6.2.2. Model Two Model two regards the rostering process as assigning nurses to serve in the eleven shifts in each day of a week. Since there may be more than one nurse working in one shift of a day, the variables take on sets of nurses as values. This kind of variables are called *constrained set variables* [31, 28]. Thus, we model each shift of a day *Shift-DayOfWeek* as constrained variables, each of which has as domains the power set of the set of all nurses. Regardless of the number of nurses in the AEU, the rostering system contains 77 (11×7) variables. If there are n nurses in the AEU, the size of the domain of each variable is 2^n. In practical constraint programming systems, elements of the domain of a set variable are not stored explicitly. In ILOG Solver library 3.0 [31], the domain is represented by two sets:

- the *possible set*—the set of those elements that belong to at least one of the possible values of the variable;
- the *required set*—the set of those elements that belong to all possible values of the variable.

In Conjunto [28, 29], the domain of a set variable is regarded as a set of ground sets, which is represented by its greatest lower bound (intersection of the ground sets) and least upper bound (union of the ground sets).

The expression of the planning rules is now based on set operations and constraints. For example, rule 1 is now expressed as the constraints:

> *For each day* `Day`*, consider the set of variables*
>
> $$V = \{\texttt{A-Day}, \texttt{P-Day}, \texttt{N-Day}, \texttt{E-Day}, \ldots, \texttt{VL-Day}, \texttt{SD-Day}, \texttt{SOL-Day}\}.$$
>
> *The following two constraints must be satisfied: (1) Elements of V are pairwise disjoint and (2) the cardinality of $\bigcup V$ is equal to the number of nurses.*

Similarly, rule 2 is expressed as:

> *Consider the set*
>
> $$V = \{\texttt{O-Mon}, \texttt{O-Tue}, \ldots, \texttt{O-Sun}\}.$$
>
> *The following two constraints must be satisfied: (1) Elements of V are pairwise disjoint and (2) the cardinality of $\bigcup V$ is equal to the number of nurses.*

Again, the soft constraints in this model are represented similarly as imperative constraints, and are posted to the solving engine using choice points.

6.2.3. Combining the Models The combined model contains model one and model two as sub-models. In addition, a set of channeling constraints are used to relate variables in the two models so that constraints can be propagated between the two models. The channeling constraints are of the following form:

$$\texttt{Nurse-DayOfWeek == Shift} \iff \texttt{Nurse} \in \texttt{Shift-DayOfWeek}.$$

It is worth noting that the constraints are generated mechanically using a simple double `for`-loop in our implementation. In general, there are $7mn$ channeling constraints for m nurses and n shift types for a weekly roster.

Variables in model one denote shifts while variables in model two denote sets of nurses. Although there are no one-one correspondences between variables in one model and values in another as in the n-queens example, the channeling constraints relate every shift to every nurse, and vice versa. Again, under this formulation, constraints in model one (or two) can be regarded as redundant constraints of the channeling constraints plus constraints in model two (or one).

6.3. Implementation and Results

To verify the correctness and effectiveness of our modeling, we have implemented the three models on personal computer (PC), which is chosen over workstation for PC's wide availability. Our prototypes consists of a user-interface and a rostering engine. The former is implemented in Microsoft Visual Basic 3.0 while the latter is realized in C++ with ILOG Solver library 3.0 [31]. Class constraints [31] of ILOG Solver facilitate our object-oriented design and greatly reduce our coding effort.

Implementations of model one, model two and the combined model consist of 6015, 4523, and 8656 lines of C++ code respectively. This little difference in code size is a result of the fact that constraint expressions occupy relatively few lines as compared to coding for I/O, data structure definition, *etc.* Therefore, much coding can be shared and combined in implementing the combined model, resulting in no significant increase in code size. *Most important of all, we implement model two and the combined model in four man-weeks, whereas model one is implemented in four man-months*. It takes longer time in the first implementation since we spend much time in problem understanding, correspondence with users, experimentation with ideas and modification. The time used for the second and third implementations is purely devoted to coding.

Our real-life benchmark data contains 27 nurses and 11 shift types. The rostering system produces weekly roster sheet[5]. The implementation of model one generates 3884 constraints, out of which 266 are imperative, during execution. Executing the model two implementation results in 1827 constraints, out of which 1675 are imperative. These unbalanced figures reflect two phenomena. First, some types of constraints are more expressive and concise than others. For example, in model one, rule 2 requires 1 constraint for each nurse, 27 in this case. In model two, the same rule requires 2 constraints irrespective of how many nurses there are. Second, some planning rules that are easier to express in one model can be tedious

Table 5. A Difficult Rostering Problem

Nurse	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Nurse-01							
Nurse-02	A	O					
Nurse-03		A	P			A	
Nurse-04							
Nurse-05		P	SD	SD			O
Nurse-06							A
Nurse-07							
Nurse-08		A		A			O
Nurse-09	VL	VL	VL	VL	VL	VL	
Nurse-10					O	P	
Nurse-11							
Nurse-12						A	O
Nurse-13	SD	SD	SD	SD	SD		O
Nurse-14			O	SD		A	
Nurse-15		A					
Nurse-16	VL	VL	VL	VL	VL	VL	
Nurse-17							
Nurse-18							
Nurse-19							
Nurse-20						O	
Nurse-21		SD	A				
Nurse-22							
Nurse-23							
Nurse-24		SD					
Nurse-25						O	
Nurse-26							
Nurse-27	A		SD	SD	A		

to formulate in another model. Finally, the combined model consists of, in addition to the constraints from models one and two, 2079 channeling constraints.

We employ the simple smallest-domain-first variable ordering heuristic and the smallest-first value ordering heuristic in the variable labeling process. In most cases, the prototypes can return a solution within 5 seconds running on a Pentium PC. In the following, we present the timing result of the three models for a particularly difficult problem instance. Table 5 shows the preset requests of this particular week's roster, which should partially explain why this problem instance is difficult. In this week's roster assignment, 17 out of the 27 nurses request for preset shifts. Among them, 5 nurses request the day-off (O) shift on Sunday, which is most wanted by other nurses as well. 2 nurses request vacation leave (VL), which extends across the entire week. Last but not least, one nurse is assigned special duty (SD) from Monday to Friday and requests for O on Sunday. As a result, 47 out of 189 slots are preset before the roster can be generated. Some of the preset slots are filled in with the most undesirable (from the rostering point of view) patterns, making this week's roster exceptionally hard to generate.

Table 6. Benchmark Results

Soft Constraints	Model 1	Model 2	Combined Model	
			Label 1	Label 2
Problem 1	short	short	short	short
Problem 2	long	long	fast	1 min
Problem 3	long	long	1 min	long
Problem 4	long	long	fast	long
Problem 5	long	long	fast	long
Problem 6	long	long	fast	long
Problem 7	long	long	long	long
Problem 8	8 min	5 min	1 min	2 min
Problem 9	1 min	long	fast	long
Problem 10	long	fast	fast	fast
Problem 11	1 min	fast	fast	fast
Problem 12	long	fast	long	fast
Problem 13	long	fast	fast	fast
Problem 14	long	long	fast	long
Problem 15	fast	fast	fast	fast

The number of preference rules posted and the nature of the preference rules posted also affect the difficulty of the problem. We vary the number and combination of preference rules posted to obtain fifteen problem instances. The results of the three models are shown in table 6. Column one of each row specifies the problem instance number. Columns two and three contain the timings of models one and two respectively. The string "short" means that execution finishes within 1 minute. The string "fast" means an almost instantaneous response and "long" means that the system does not return an answer within 30 minutes. Since the combined model contains models one and two as sub-models, labeling variables of either model will instantiate also variables in the other model. Column four shows the timing result of the combined model by labeling only variables in model one. Alternatively, we can label only variables in model two and yield the results in column five. Experimental results confirm that the combined model exhibits significant speedup over either model one or two.

The execution of the nurse rostering system can be divided into several phases. First, all preset requests are collected and told to the constraint store as simple equality constraints. Second, all hard constraints are generated and told to the constraint store. At this stage, some domain values are pruned because they are incompatible with the preset requests and hard constraints. We confirm from our tracing that more pruning is achieved in the combined model implementation than the implementations based on only single models. Third, soft constraints in the form of disjunctions are also added to the constraint store. Further pruning takes place. This time, incompatible branches are pruned. Again, we observe more pruning in the combined model. The last stage of execution is the search phase, in which variables are labeled one by one using variable/value ordering heuristics. The

extra pruning achieved in the combined model enables searching to start off closer to the solution branch than the starting position of the single models. Suppose we search depth-first from left to right. In other words, the starting position of the combined model is to the "right" of those of the single models. During variable labeling, more constraint propagation in the combined model obtains earlier failure and quicker descent to the solution branch. The situation is similar to the one depicted in the 9-queens example in section 5.

7. Concluding Remarks

7.1. Summary of Contributions

The contribution of this paper is five-fold. First, we present the notions of model and model redundancy of constraint satisfaction problems. Second, we introduce channeling constraints for combining mutually redundant models of the same problem. We further suggest guidelines for constructing alternate models for a problem and for how the models can be connected by a form of channeling constraints. In the nurse rostering application, the channeling constraints are generated mechanically, if not automatically, using a simple double `for`-loop. Third, benchmark results from the n-queens case study demonstrate the extra pruning attained from redundant modeling in terms of the number of failures and choice points. In the same case study, we show that redundant modeling can integrate seamlessly with other heuristic techniques. Fourth, Jourdan's new value ordering heuristic [35] suggests that redundant modeling opens up new horizon for the definition of new value ordering heuristics. Fifth, we apply our method in a real-life nurse rostering problem and verify empirically that the combined model does exhibit significant speedup over either of the mutually redundant models.

7.2. Discussions

The two case studies provide useful insights into the redundant modeling approach. With more constraints and variables in a combined model, time overhead is unavoidable. We do not advocate the use of redundant modeling to tackle every type of CSP. The technique is beneficial only when the gain from extra pruning surpasses the overhead. We conjecture the class of problems that can benefit most from redundant modeling to require large amount of searching or exhibit poor thrashing behavior [42].

A potential problem of our method is the memory overhead introduced by the accommodation of more than one model (variables and constraints) of a problem. There is always a tradeoff between time and space in the design of algorithms. The choice is always dictated by either limitation on hardware or urgency for time-efficiency. With the advent of cheaper and more massive memory, we do not foresee memory overhead as a major obstacle for the adoption of our method.

7.3. Future Work

This paper reports our first results and experience in using the redundant modeling method. Some open problems remain. We definitely need more experience in applying the method to other real-life problems. In particular, we should investigate automatic or semi-automatic method of creating alternate models from an existing model. Such exercise is useful even if we can automate model generation for only problems of a restricted problem domain. Another open problem is the code maintenance of the combined model: how modifications in one model can be "propagated" to the other models automatically. To solve these problems, we may benefit from a formal model description language or notation so that models can be described fully and formally. Such a language helps in extracting and formally reasoning with properties of models.

On the empirical side, it is also interesting to study and experiment with variable ordering heuristics that label variables in the sub-models alternatively.

Wallace [54] suggests that each sub-model in a combined model does not have to be a complete model of the problem at hand. We call such partial models *viewpoints*. The idea is to represent each constraint of a problem in a viewpoint that allows the most succinct expression of the constraint and/or gives the best pruning behavior to the constraint. These viewpoints may share redundant information but a combined model formed by connecting these viewpoints using channeling constraints must give a complete characterization of the problem being tackled. This approach incurs less memory and time overhead than redundant modeling. However, more guidelines are needed for determining if and when a viewpoint is "suitable" for a constraint. We may be able to benefit from the work of Eaton and Freuder [19]. It is interesting to redo the design and implementation of the nurse rostering application in order to gain more experience of this approach.

Acknowledgments

We are indebted to Mr. Vincent Tam of the Tang Shiu Kin Hospital, who spared great patience in explaining to us the planning rules and rostering techniques. Our work benefited immensely from many fruitful discussions with Mr. Tam and his often timely response to our queries. Constructive comments from the anonymous referees from CP96 and IEEE ICTAI'96 helped improve the quality of the final version of the paper. The critical comments from the anonymous reviewers and Editor Boi Faltings of the CONSTRAINTS journal injected many useful ideas into the technical content and writing style of the paper. Editor-in-Chief Eugene Freuder provided continuous encouragement in addition to suggestions for improvement of the paper and pointers to related work. We are also grateful to the discussions with François Fages during CP96 and his pointer to the new value ordering heuristic for the n-queens problem in Jean Jourdan's doctoral thesis. We thank Jean-François Puget and Michel Gendreau for pointing out related work. Ken McAloon provided pointers to related work and a new ILOG Solver feature suitable for expressing channeling constraints. Henry Ho helped to perform some last minute implemen-

tation work. Last but not least, this paper would not have come into existence without Ho-Fung Leung's initial confirmation that the redundant modeling idea is indeed new.

Notes

1. Given a set of constraints C, a constraint c is *redundant* with respect to C if C entails c.
2. Caseau and Koppstein [9] use the terminology "strategies" instead of "models." A *strategy* is defined by three components: (1) identification of the incomplete information on which the solver will work, (2) statement of the constraints, and (3) definition of additional information and propagation rules to maintain the strategy.
3. In some applications, not all machines need to be running at all time to complete all jobs on time. The dual view of the problem can be maintained by assigning artificial dummy jobs to idling machines.
4. Two of the ten preference rules cannot be expressed as constraints. They are implemented algorithmically. The remaining eight rules are numbered C2.1, C2.2, C2.3a, C2.5, C2.6, C2.7, C2.8, and C2.9 respectively. Interested readers are referred to [10, 12] for details.
5. Generating more than one week's roster can be achieved by running the system multiple times.

References

1. A. Aggoun. A good value-ordering heuristics for the n-queens problem. Private Communication, 1994.
2. P. Baptiste and C. Le Pape. Disjunctive constraints for manufacturing scheduling: Principles and extensions. In *Proceedings of the Third International Conference on Computer Integrated Manufacturing*, Singapore, 1995.
3. N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling*, 20(12):97–123, 1994.
4. B.W. Benson Jr. and E.C. Freuder. Interchangeability preprocessing can improve forward checking search. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 28–30, 1992.
5. C. Bessière and M. Cordier. Arc-consistency and arc-consistency again. In *Proceedings of AAAI-93*, pages 108–113, 1993.
6. J. Bitner and E.M. Reingold. Backtrack programming techniques. *Communications of the ACM*, 18:651–655, 1985.
7. Y. Caseau. A formal system for producing demons from rules. In *Proceedings of the 1989 International Conference on Deductive and Object-Oriented Databases*, pages 203–219, 1989.
8. Y. Caseau. An object-oriented deductive language. *Annals of Mathematics and Artificial Intelligence*, 1991.
9. Y. Caseau and P. Koppstein. A rule-based approach to a time-constrained traveling salesman problem. Unpublished manuscript, 1992.
10. 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 Eighth IEEE International Conference on Tools with Artificial Intelligence*, pages 140–148, November 1996.
11. B.M.W. Cheng, J.H.M. Lee, and J.C.K. Wu. Speeding up constraint propagation by redundant modeling. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming*, pages 91–103, 1996.
12. 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.
13. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

14. R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 38:1–38, 1988.
15. Y. Deville and P. Van Hentenryck. An efficient arc consistency algorithm for a class of CSP algorithm. In *Proceedings of IJCAI 1991*, pages 325–330, 1991.
16. M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In *Proceedings of the European Conference on Artificial Intelligence*, pages 290–295, 1988.
17. M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, and T. Graf. Applications of CHIP to industrial and engineering problems. In *Proceedings of the 1st International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*, pages 887–892, 1988.
18. M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'88)*, pages 693–702, Tokyo, Japan, December 1988.
19. P.S. Eaton and E.C. Freuder. Agent cooperation can compensate for agent ignorance in constraint satisfaction. In *Proceedings of the 1996 AAAI Workshop on Agent Modeling*, pages 24–29. AAAI Press, August 1996. Technical Report WS–96–02.
20. T. Ellman. Abstraction via approximate symmetry. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 916–921, 1993.
21. E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29:24–32, 1982.
22. E.C. Freuder and D. Sabin. Interchangeability supports abstraction and reformation for constraint satisfaction. In *Proceedings of the Symposium on Abstraction, Reformulation and Approximation*, 1995.
23. J. Gaschnig. A general backtrack algorithm that eliminates most redundant tests. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 1977.
24. J. Gaschnig. Experimental case studies of backtrack versus Waltz-type versus new algorithms for satisfying assignment problems. In *Proceedings of the Second Biennial Conference of the Canadian Society for Computational Studies of Intelligence*. Canadian Information Processing Society, 1978.
25. J. Gaschnig. *Performance Measurement and Analysis of Certain Search Algorithms*. PhD thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, U.S.A., 1979.
26. J. Gaschnig. Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Carnegie-Mellon University, 1979.
27. P.A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problem. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 31–35, 1992.
28. C. Gervet. Conjunto: Constraint logic programming with finite set domains. In *Logic Programming: Proceedings of the 1994 International Symposium*, pages 339–358, 1994.
29. C. Gervet. Interval propagation to reason about sets: definition and implementation of a practical language. *CONSTRAINTS*, 1(3):191–244, 1997.
30. R.M. Haralick and G.L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
31. ILOG. *ILOG SOLVER: Reference Manual Version 3.0*, 1995.
32. ILOG. *ILOG SOLVER: User Manual Version 3.0*, 1995.
33. ILOG. *ILOG SOLVER: Reference Manual Version 4.0*, 1997.
34. J. Jourdan. Concurrent constraint multiple models in CLP and CC languages: toward a programming methodology by modelling. In *Proceedings of the INFORMS Conference*, New Orleans, USA, 1995.
35. J. Jourdan. *Concurrent constraint multiple models in CLP and CC languages: Toward a programming methodology by modelling*. PhD thesis, Denis Diderot University, Paris VII, February 1995.
36. T. Kökény. Constraint satisfaction problems with order-sorted domains. *International Journal on Artificial Intelligence Tools*, 4(1&2):55–72, 1995.
37. V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13:32–44, 1992.

38. C. Le Pape. Implementation of resource constraints in ILOG SCHEDULE: A library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering*, 3:55–66, 1994.
39. C. Le Pape. Resource constraints in a library for constraint-based scheduling. In *Proceedings of the INRIA/IEEE Conference on Emerging Technologies and Factory Automation*, Paris, France, 1995.
40. J.H.M. Lee, H.F. Leung, P. Stuckey, V. Tam, and H.W. Won. Using stochastic methods to guide search in CLP: a preliminary report. In *Proceedings of the 1996 Asian Computing Science Conference*, pages 43–52, 1996.
41. J. Liu and K. Sycara. Emergent constraint satisfaction through multi-agent coordinated interaction. In *Proceedings of MAAMAW'93*, Neuchatel, Switzerland, 1993.
42. A.K. Mackworth. Consistency in networks of relations. *AI Journal*, 8(1):99–118, 1977.
43. A.K. Mackworth, J.A. Mulder, and W.S. Havens. Hierarchical arc consistency: Exploiting structured domains in constraint satisfaction problems. *Computational Intelligence*, 1:118–126, 1985.
44. P. Marti and M. Rueher. A distributed cooperating constraints solving system. *International Journal on Artificial Intelligence Tools*, 4(1&2):93–113, 1995.
45. S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling. *Artificial Intelligence*, 58:161–205, 1992.
46. R. Mohr and T.C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.
47. B.A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989.
48. M. Perrett. Using constraint logic programming techniques in container port planning. *ICL Technical Journal*, 7(3):537–545, May 1991.
49. P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.
50. E.P.K. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.
51. P. van Beek. On the minimality and decomposability of constraint networks. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 447–452, 1992.
52. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* The MIT Press, 1989.
53. P. Van Hentenryck, Y. Deville, and C.M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.
54. R. Wallace. Partial models in redundant modeling. Private Communication at CP96, 1996.
55. D. Waltz. Understanding line drawings of scenes with shadows. In P.H. Winston, editor, *The Psychology of Computer Vision*, pages 19–91. McGraw-Hill, 1975.