Speeding Up Constraint Propagation By Redundant Modeling

B.M.W. Cheng, J.H.M. Lee, and J.C.K. Wu

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
Email: {mwcheng,jlee,ckwu}@cs.cuhk.edu.hk
Fax: (+852) 26035024

Abstract. The paper describes a simple modeling and programming approach for speeding up constraint propagation. The idea, although similar to redundant constraints, is based on the concept of redundant modeling. We define CSP model and model redundancy formally, and show how mutually redundant models can be combined and connected using channeling constraints. The combined model contains the original but 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 apply our method to 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, Nurse Rostering

1 Introduction

The problem at hand is that of *constraint satisfaction problems* (CSP) defined in the sense of Mackworth [16], 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.

CSP's are, in general, NP-complete and some are even NP-hard [5]. Thus, a general algorithm designed to solve any CSP will necessarily require exponential time in problem size in the worst case. One widely-adopted approach in solving CSP's features various degrees of combinations of backtracking tree search and constraint propagation [19, 12]. This framework is realized in the constraint logic programming languages CHIP [8] and C++ class library ILOG SOLVER [11], which have been applied successfully to real-life industrial applications [6, 7, 20, 14, 1, 13].

There are at least four approaches to increase the efficiency of constraint-solving. The first approach amounts to incorporating variable- and value-ordering heuristics [3, 10] in the labeling procedure of a propagation-based constraint solver. Incorporation 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 is never formally documented or studied but one can find it used in many places in the literature, such as [22, 2]. 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 is to replace propagation-based techniques, which are algorithmic in nature, entirely by the promising stochastic techniques such as the iterative repair method [18, 15]. The fourth approach employs cooperation among different constraint solvers [17]. The last two approaches call for technological innovations, which are usually difficult to come up with.

Propagation-based constraint languages and tools are widely available. The aim of this research project is to adopt existing technology and devise a general and systematic methodology for speeding up solution search in CSP's. 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. In addition, some constraints propagate and prune well when expressed in one model and yet expression of the same constraints in another model exhibits opposite pruning behaviour.

Our approach is inspired by the effectiveness of redundant constraints[1], which are redundant semantically but augment the propagation and pruning of the original constraints of a CSP, in speeding up constraint-propagation. Intuitively speaking, our method amounts to implementing more than one model of a given problem and somehow 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 demonstrate the feasibility of our proposal, we present a case study of our method

[1] Given a set of constraints C. A constraint c is *redundant* with respect to C if C entails c.

as applied in the implementation of a working nurse rostering system for the emergency ward of a local hospital.

The rest of the paper is organized as follows. Section 2 lays down the foundation of modeling and model redundancy. Channeling constraints are introduced. The section ends by giving guidelines for generating redundant models from existing models and connecting these models. In section 3, we provide the domain description of our case study. Section 4 details two alternative modeling of the nurse rostering system and describes how the models can be connected using channeling constraints. In section 5, we present and discuss our implementation experience of the three models and some preliminary results. Section 6 summarizes our contribution and sheds light on future direction of research.

2 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[2]. 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

1. M_1 and M_2 are models of P respectively, and
2. $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$.

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 $c \notin \mathcal{C}_{\mathcal{X}_2}$. Given a problem P and a set $\{M_1, \ldots, M_n\}$ $(n > 1)$ of mutually redundant models of P. A *combined model* $M_{\{M_1,\ldots,M_n\}}$ is obtained by implementing the set of models $\{M_1, \ldots, M_n\}$ $(n > 1)$ of P and connecting these models with channeling constraints.

The redundant modeling method as presented is general and flexible. Some guidelines in its usage are in order. A first question is how many redundant models should be used. Given 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 theoretically since the individual models are embedded in the combined model. Therefore, it seems reasonable to combine and implement

[2] 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 artificially dummy jobs to idling machines.

as many mutually redundant models as one can dream up. 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.

Another common and important question to ask is how to generate a redundant model. We observe that most 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. We can find more examples of this types. 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 not on suggesting multiple perspectives on the same problem. The novelty is on suggesting to connect these perspectives to form a combined perspective which exhibits more efficient pruning behaviour.

Using reciprocal views of problems is just 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 [21] suggests two ways of modeling the 8-queens problem. The first and familiar model consists of eight variables, each of which denotes the position of a queen in a different row of the chess board. Every domain contains the eight possible positions of a queen. The 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.

Channeling constraints for models M_1 and M_2 must be able to propagate constraints from M_1 to M_2 and vice versa. 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. An effective channeling constraint is 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.

Note that redundant constraints and redundant models are orthogonal concepts although their working principles are similar. Given 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 a problem P. The constraints $\mathcal{C}_{\mathcal{X}_1}$ and $\mathcal{C}_{\mathcal{X}_2}$ have no relationship to each other at all since they do not even share variables. 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 $\mathcal{C}_{\mathcal{X}_2}$).

3 A Case Study

Our task is to design and implement a nursing staff rostering system for the Ambulance 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 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 fair rotation of want and unwanted shifts and soft constraints are referred to [4].

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 figure 1. 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.*

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
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Fig. 1. A Sample Roster Duty Sheet

Rule 2 *Each staff get 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 destroy 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 [22], 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[3]. 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 accept two consecutive A or P shift. 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.*

4 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

[3] 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 [4] for details.

shifts to nurses. Implementation of the first model performs well in general but fails to return answer in a timely manner 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.

4.1 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 [11] 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.

4.2 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* [11, 9]. 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.

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 is represented similarly as imperative constraints, and are posted to the solving engine using choice points.

4.3 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:

`Nurse-DayOfWeek == Shift` *if and only if* `Nurse` $\in$ `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.

5 Implementation and Preliminary 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 [11]. Class constraints [11] 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[4]. 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. Second, some planning rules that are easier to express in one model can be tedious 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 heuristics and the smallest-first value-ordering heuristics in the variable labeling process. In most case, 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. Figure 2 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.

The number of preference rules posted and the nature of the preference rules posted also affects the difficulty of the problem. For the same set of preset requests, we vary the combination of preference rules posted and show the performance of the three models in figure 3. Column one of each row specifies the combination of preference rules imposed. 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 mins. 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.

[4] Generating more than one week's roster can be achieved by running the system multiple times.

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		

Fig. 2. A Difficult Rostering Problem

6 Concluding Remarks

The contribution of this paper is three-fold. First, we define formally the notions of modeling and model redundancy of constraint satisfaction problems. These definitions serve as the foundation for the systematic study of the use of redundant modeling to speed up constraint-solving. 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

Soft Constraints	Model One	Model Two	Combined Model: One Vars	Combined Model: Two Vars
—	short	short	short	short
C2.1+C2.3a	long	long	fast	1 min
C2.1+C2.2	long	long	1 min	long
C2.1+C2.2+C2.3a	long	long	fast	long
C2.1+C2.2+C2.3a+C2.6	long	long	fast	long
C2.1+C2.3a+C2.6	long	long	fast	long
C2.1+C2.6	long	long	long	long
C2.1	8 min	5 min	1 min	2 min
C2.2	1 min	long	fast	long
C2.3a	long	fast	fast	fast
C2.5	1 min	fast	fast	fast
C2.6	long	fast	long	fast
C2.7	long	fast	fast	fast
C2.8	long	long	fast	long
C2.9	fast	fast	fast	fast

Fig. 3. Benchmark Results

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, 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.

We believe that the redundant modeling method will be a valuable tool for the constraint community. The method, although simple, is systematic enough, allowing programmers to generate channeling constraints almost mechanically. Our approach also requires no special insight into the problem domain or the problem itself. This is not true for the case of introducing redundant constraints. Redundant models and channeling constraints are different from redundant constraint although they are based on a similar working principle. Programmers can still exploit full knowledge of the problem or the problem domain to inject redundant constraints into one or all of the redundant models to further speed up constraint-solving. Another nice property of our method is that it implies no modification to the underlying constraint solver or labeling heuristics.

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.

Results in this paper are preliminary. Many 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. Another important direction of work is to understand and analyze the propagation behaviour of constraints among sub-models in a combined model. One conjecture is that propagation among sub-models achieves a higher consistency level of the combined network.

Acknowledgement

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 benefitted immensely from many fruitful discussions with Mr. Tam and his often timely response to our queries. Constructive comments from the anonymous referees help improve the quality of the final version of the paper. Last but not least, this paper would not have come into existence without Ho-Fung Leung's confirmation that the redundant modeling idea is indeed new.

References

1. 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.
2. N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling*, 20(12):97–123, 1994.
3. J. Bitner and E.M. Reingold. Backtrack programming techniques. *Communications of the ACM*, 18:651–655, 1985.
4. B.M.W. Cheng, J.H.M. Lee, and J.C.K. Wu. A constraint-based nurse rostering system using a redundant modeling approach. Technical report (published), Department of Computer Science and Engineering, The Chinese University of Hong Kong, 1996.
5. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

6. 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.
7. 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.
8. 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.
9. C. Gervet. Conjunto: Constraint logic programming with finite set domains. In *Logic Programming: Proceedings of the 1994 International Symposium*, pages 339–358, 1994.
10. R.M. Haralick and G.L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
11. ILOG. *ILOG: Solver Reference Manual Version 3.0*, 1995.
12. V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13:32–44, 1992.
13. 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.
14. 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.
15. J. Liu and K. Sycara. Emergent constraint satisfaction through multi-agent coordinated interaction. In *Proceedings of MAAMAW'93*, Neuchatel, Switzerland, 1993.
16. A.K. Mackworth. Consistency in networks of relations. *AI Journal*, 8(1):99–118, 1977.
17. P. Marti and M. Rueher. A distributed cooperating constraints solving system. *International Journal on Artificial Intelligence Tools*, 4(1&2):93–113, 1995.
18. 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.
19. B.A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989.
20. M. Perrett. Using constraint logic programming techniques in container port planning. *ICL Technical Journal*, 7(3):537–545, May 1991.
21. E.P.K. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.
22. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* The MIT Press, 1989.

This article was processed using the LaTeX macro package with LLNCS style