A Constraint-Based Nurse Rostering System Using a Redundant Modeling Approach

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

Abstract

This paper describes the design and implementation of a nurse rostering system using a redundant modeling approach. Nurse rostering is defined as a process of generating timetables for specifying the work shifts of nurses over a given period of time. This process is difficult because the human roster planner has to ensure that every rostering decision made complies with a mixture of hard hospital rules and soft nurse preference rules. Moreover, some nurse shift pre-assignments often break the regularity of wanted (or unwanted) shifts and reduce the choices for other unfilled slots. Soft constraints amount to disjunction, which can be modeled as choices in the search tree. This approach, although straightforward, incurs overhead in the search of solution. We propose redundant modeling, an effective way to speed up constraint propagation through cooperations among different models for the same problem, as a means to reduce search time. Experiments and pilot testing of the system confirm the feasibility of our method.

1. Introduction

Many real-world organizations such as hospital, hotel and paging operation center have their staff work in shift patterns. In some cases, each worker is assigned a permanent shift; but in others, staff members work according to rosters that change periodically. The process of determining the shifts of staff members over a fixed period of time is called *staff rostering*. Many factors are involved in this planning process: imperative organization rules, staff preference requests, anticipated (or unanticipated) staff availability, *etc*.

Staff rostering are usually done by human duty planners. However, the complexity of planning rules and consideration of various factors make roster generations error-prone and inefficient. Therefore duty planners often resort to computer tools to aid their planning process. Most existing tools provide only a user-interface for manual roster generation and various facilities to check the validity of the prepared rosters. Some more advanced systems incorporate both the roster checking process and roster generation process. Duty planners only need to inform these systems factual information like specific staff shift requests, and let the systems do the rest. We call such a system a staff rostering system.

The problem at hand is to design and implement a nurse rostering system in a real world setting. The participating hospital is Tang Shiu Kin Hospital, Hong Kong. The problem domain involves generating weekly timetables for a group of nurses, whose ranks, rules to be abided (be it hard or soft) and specific shift requests are different. The duty planner currently produces the duty roster manually.

This task is difficult for the duty planner for two reasons. First, there are a number of hard hospital rules and soft nurse preference rules to be obeyed. Although there are relatively few nurses (about 25-28 nurses), most of the hard rules are very tight, and many soft planning rules are often incompatible with each other. Second, the duty planner allows nurse to request shift pre-assignments. The requests often make the roster generation process even more difficult. For example, suppose a group of nurses request to have a day-off on Sunday. If this group is too large, it may over-reduce the available manpower to work on Sunday. It may also disturb normal sequences of some wanted (or unwanted) shifts and affect the fairness and quality of the rosters. In addition to assigning who to work on Sunday, the duty planner also has to consider whose shift requests to reject.

We employ a constraint-based approach in the construction of this nurse rostering system. This approach amounts to modeling the nurse rostering problem as a constraint satisfaction problem and using constraint programming tools for system implementation. A constraint satisfaction problem (CSP) is defined as [7]:

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.

After modeling the problem as a CSP, constraint programming tools are used to build up the constraint-solving engine. Constraint programming tools provides a rich set of constraint-solving functions and routines, and greatly facilitates the implementation process. The expressiveness and efficiency of constraint programming in solving such kind of scheduling problem are described in [11]. In this project, we adopt ILOG SOLVER, a C++ class library [12, 4], as our tool.

We encounter two major difficulties in the development process. First, shift pre-assignment requests adversely affect the efficiency of the rostering system in two ways. They cause the choices for remaining shifts to become tight and limited, and break the regularity of some wanted and unwanted shifts. Second, the problem domain consists of a series of soft planning rules. We model soft rules as disjunctions in the search tree. In this approach, the search space increases combinatorially with the number of soft rules and nurses. Solution generation efficiency will be degraded. We cope with this problem by applying the concept of redundant modeling [2] in the system. We create two different models for the nurse rostering problem. By creating a model for a problem, we mean the process of determining the variables, the associated domains for the variables, and the expressions of the constraints. In the first model, the variables are nurses and the domains are the shifts; in the second model, the variables are the shifts and the domains are the nurses. The two models are then connected using channeling constraints to speed up constraint-solving [2].

The rest of this paper is organized as follows. Section 2 provides an account of related work in the field of nursing staff rostering. In section 3, the domain of our rostering problem is detailed. Operations and structures of the Tang Shiu Kin Hospital, as well as the various planning rules, are described in length. Section 4 describes the modeling and implementation details of the nurse rostering system, followed by a discussion of some benchmarking results in Section 5. Section 6 summarizes our contributions and sheds light on future research direction.

2. Related Work

There have been extensive attempts to solve the nurse rostering problem [13, 14, 5, 17, 6, 10, 16, 9, 8, 1], which can be classified into three main approaches. The first approach amounts to modeling the rostering problem mathematically and applying methods in operations research to search for solutions [16, 9]. This approach, though a solution can be found in a relatively short time, is too rigid to incorporate the flexibility of the problem domain. It also requires a lot of computing resources [8]. The second approach is to design domain-specific heuristics and algorithms for solving the rostering problem [10, 8, 1]. It usually formulates the problem as optimization of certain criteria, such as nurse satisfaction and staff turn-over percentage. These methods are not generic. In addition, successful algorithms and heuristics in a particular scenario may not perform well in other problem domains [11]. Moreover, these systems are often written in imperative languages, which cast large implementation overhead. The last approach employs constraint programming [17, 11, 6]. This approach is by far the most promising for its expressiveness to model problems as CSP's, in which one only needs to concentrate on the representation of variables and constraints. Solutions can be transparently generated by built-in solution generation procedure in constraint programming languages. Slight modification to the problem domain usually induces small change to the implementation. The emergence of various constraint programming tools further allow other programming methodologies, such as object-orientation, to be incorporated in constraint programming.

Our problem instance differs from those in the literature in three aspects. First, nurses in our domain prefer to work for different shifts in different days. That means that the duty planner needs to generate non-consecutive shifts patterns. This process is difficult to achieve using specific algorithms, and constraint programming facilitates expression of such requirements. On the other hand, it is relatively simple to design specific algorithms to generate consecutive shifts over consecutive days [6, 8]. Second, in most of the existing nurse rostering problems, only a few main shifts (AM, PM, Night, Day-off) are involved. However, our problem domain has a total of eleven shifts, and many of them are governed by complicated planning rules. This increases the difficulty and complexity of our problem considerably. Third, rosters are not generated as cyclic schedules, in which nurses take turn in performing each and every one of the shift patterns in the roster. There are two implications. Rosters have to be generated every week, increasing the workload of the duty planner. Fairness in shift assignment, which is one of the dominant factors that determines the morale of the nursing staff, becomes an important criterion in roster generation.

3. Domain Description

Tang Shiu Kin Hospital a multi-discipline hospital in Hong Kong. It consists of a number of operation units, the Ambulance and Emergency Unit (AEU) being one of them. The AEU provides 24-hour emergency services to the general public seven days a week.

Currently, there are fifty five personnels at the AEU. Twenty-eight of them are nurses of different ranks. These ranks include midwife nurses (NM), registered nurses (RN), enrolled nurses (EN), male registered nurses (RN(m)), enrolled nurses (EN(m)) and part-time nurses (PTN).

Nurses in the AEU work in shifts. Generally speaking, 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 nursing staff member. Nurses are entitled to different types of holiday. These includes 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). Totally there are eleven shift types.

There are several steps for the duty planner in the planning process. The first step is to collect information about pre-assigned shifts and shift requests for the nurses. Then the duty planner generates the remaining shift slots, observing all planning rules and nurse preference rules. The last step is finalizing the timetable by making modifications to accommodate any further nurse requests and to remove inconsistency and human error.

The planning rules to be respected by the duty planner can be categorized into two main types: 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.

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

Rule 2 *Each staff gets one O shift per week.*

Rule 3 *If a nurse is assigned an N shift on a particular day, the nurse must be assigned an A shift in the previous day.*

Rule 4 *A VL shift should start on a Monday, and should be taken in a multiple of six days. The minimum number of VL is six days. Every six VL shifts must be followed by an O shift on Sunday.*

Rule 5 *For each day, there should be eight nurses working in the A shift, six nurses working in the P shift, two nurses working in the E shift, and three nurses working in the N Shift.*

Rule 6 *There are some requirement on the number of a particular type of nurse working in a particular shift. They are summarized as follows:*

Type of shift	# Male nurses	# N/M	# EN
A	≤ 3	≥ 1	
P	≤ 3	≥ 1	
N	≤ 1	= 1	≤ 1

Preference planning rules should be satisfied as much as possible but they can be violated. Since these planning rules are mainly nurse preferences on the timetable, they define the "goodness" of the timetable generated. The more preference planning rules a timetable respects, the better the timetable is.

Rule 7 *A nurse should not work for the same shift for two consecutive days. In particular, a nurse prefers A shifts alternating with P/E shifts. If such an arrangement is impossible, a nurse also accepts two consecutive A or P shifts. However, assigning three consecutive A or P/E shifts to a nurse is strongly discouraged.*

Rule 8 *Apart from all holiday schedule, the number of A shifts and P shifts allocated to each nurse each week should be roughly balanced.*

Rule 9 *N shifts are unwanted. Nurses should take turn to be assigned an N shift. To be exact, if there are Y nurses of a particular rank to be scheduled for N shift, each nurse in that group should have an N shift every Y days (refer to Rule 6).*

Rule 10 *Immediately after an N shift, a nurse prefers to take an O shift. If it is not possible, a nurse prefers to take a P shift. If it is not possible, a nurse accepts an A shift.*

Rule 11 *E shift allocation is similar to Rule 9. The difference is that the AEU requires two nurses working in the E shift everyday, while three nurses are required for the N shift.*

Rule 12 *Generally, if a nurse has already performed three or four N shifts, the first compensation-off (CO1) can be allocated to the nurse. Similarly, if a nurse has already performed six or seven N shifts, the second compensation-off (CO2) can be allocated.*

Rule 13 *Immediately after a holiday (O, CO, PH), a nurse prefers to have a P shift.*

Rule 14 *Immediately after a VL shift, a nurse starts to work on a Monday. The nurse prefers to work in the P shift on Monday, A shift on Tuesday and N shift on Wednesday.*

Rule 15 *If possible, a nurse prefers to have consecutive holidays.*

Rule 16 *A nurse prefers to work with the following patterns.*

Preferred consecutive patterns
A $\rightarrow$ O $\rightarrow$ P
A $\rightarrow$ N $\rightarrow$ O $\rightarrow$ P

Rule 17 *Each nurse should have equal opportunity to get O shifts on Saturdays and Sundays, and PH shifts on bank holidays.*

4. Modeling and Implementation

This section describes the modeling and implementation of the system. Our system consists of two separate programs. The first program is a core constraint engine for generating rosters. This core engine is built using the principle of redundant modeling [2]. This concept is to define and connect multiple, yet mutually redundant, CSP models for the same problem to form an aggregate model. The aggregate model exhibits significant speedup in constraint propagation over any of the model when used alone. The second program is a graphical user interface for user interaction. In the following, we give a brief introduction to redundant modeling, followed by a description of the two models for our nurse rostering problem.

4.1. Principle of Redundant Modeling

We can view virtually any real-world scheduling process in two ways. For example, a generic job-shop scheduling problem can be formulated as assigning machines to workers or assigning workers to machines. Similarly, a school timetabling problem can be seen as allocation of courses to timeslots or allocation of timeslots to courses.

From a CSP point of view, each modeling approach corresponds to one CSP and a constraint network. There is usually not a single model that is effective enough to tackle real world problems efficiently[1]. Experience suggests 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 behavior.

Our approach is inspired by the effectiveness of redundant constraints[2], 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*. Suppose now we have two models for the same problem. The first model assigns type X to type Y and the second model assigns type Y to type X. An effective channeling constraint in this case takes the following 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.

4.2. Problem Model

We propose two models for the nurse rostering problem. The first is to allocate shifts to nurses, and the second is to assign nurses to shifts.

Slot Model The slot model for the nurse rostering problem is based on the layout of the roster, as shown in Figure 1, which suggests allocating shifts to nurses. Each slot in the roster constitutes a variable `Nurse-Day` of the model, which denotes the shift assigned to nurse `Nurse` on the day `Day`. We call each such variable a *slot variable*. The associated domains of slot variables contain the shifts available to the nurses. As the roster is generated on a weekly basis, each nurse is associated with seven slot variables.

Shift Model This model results from another view of the problem. The shifts become variables. We call them *shift variables*. Each shift variable `Shift-Day` denotes the set of nurses assigned to work in shift `Shift` on the day `Day`. There are usually more than one nurse working in a shift so that the domain of each shift variable is the powerset of the set of all nurses. The scenario is depicted in Figure 2. For a weekly roster, the number of such variables is seven times the number of shift types, and is constant regardless of the number of nurses. However, the size of the domains depends on the number of shift types.

Aggregate Model The two models are connected to form an aggregate model of the problem, using constraints of the following form:

$$\texttt{Nurse-Day == Shift} \Leftrightarrow \texttt{Nurse} \in \texttt{Shift-Day}$$

[1] An effective model of the problem should facilitate convenient expression of constraints, efficient pruning and efficient constraint-solving.

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

Nurse	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Nurse-01	A	E	N	P	O	SD	P
Nurse-02	P	O	A	A	CO1	PH	A
Nurse-03	SD	SD	SD	O	A	N	P

Figure 1. First view of the problem

Shift	Mon	Tue	Wed	Thu	Fri	Sat	Sun
A shift	N-1, N-2 N-3	N-4, N-5 N-6	N-7, N-8 N-9	N-10 N-11	N-12 N-13	N-14 N-15	N-16 N-17
P shift	N-10 N-11	N-12 N-13	N-14 N-15	N-16, N17 N-18	N-1, N-2 N-3	N-4, N-5 N-6,	N-7, N-8 N-9
O shift	N-4	N-8	N-12	N-15	N-5	N-7	N-10

Figure 2. Second view of the problem

for each nurse `Nurse`, each day `Day`, and each shift `Shift`.

4.3. Hard Constraints Modeling

Hard constraints are those imperative planning rules which must be satisfied at all cost. These imperative rules can be expressed conveniently as constraints since constraints are by default hard in most constraint programming tools. In the following, we discuss how we model only some of the representative constraints in the two models, given the length limitation of the paper.

Slot Model In the slot model, 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 [4] as follows:

> *For each nurse* `Nurse`*, the number of variables in* $\{\texttt{Nurse-Mon}, \texttt{Nurse-Tue}, \ldots, \texttt{Nurse-Sun}\}$ *assigned with the O shift must be equal to one.*

Rule 3 is expressed using a boolean implication constraint $\Rightarrow$ [4], which is a reflection of a boolean implication logical relation between two variables, as follows:

> *For each nurse* `Nurse` *and for each* `Day` *from Tuesday to Sunday,* `Nurse-Day` *is N* $\Rightarrow$ `Nurse-Day'` *is A, where* `Day'` = `prev(Day)`*, and* `prev()` *is a function that takes a day as input and returns the previous day of the input.*

Shift Model The variables in this model are constrained set variables [4]. As a result, the planning rules are modeled using primitive set constraints [4] on these variables. Rule 1 is expressed using two set constraints on the eleven shifts in a day:

> *For each day* `Day`*, consider the set* $V = \{\texttt{A-Day}, \texttt{P-Day}, \texttt{N-Day}, \ldots, \texttt{SOL-Day}\}$*. 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 can be expressed as follows:

> *Consider the set* $V = \{\texttt{O-Mon}, \ldots, \texttt{O-Sun}\}$*. 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.*

Since the shift variable `O-Day` denotes the set of nurses having an O shift on day `Day`, the first constraint above ensures that no nurse gets more than one O shift per week. The union of all `O-Day` shift variables for every day of a week represents the set of nurses who get O-shift in the week. Constraining the cardinality of the union ensures that every nurse gets an O shift in a week. The expression of Rule 3 in this model uses the subset constraint as follows.

> *Consider the sets* $A = \{\texttt{A-Mon}, \ldots, \texttt{A-Sun}\}$ *and* $N = \{\texttt{N-Mon}, \ldots, \texttt{N-Sun}\}$*. For each* `Day` *from Tuesday to Sunday, we have* `N-Day` $\subseteq$ `A-Day'`*, where* `Day'` = `prev(Day)`*, and* `prev()` *is a function that takes a day as input and returns the previous day of the input.*

4.4. Soft Constraints Modeling

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.

Some of the soft constraints are mutually incompatible. In other words, satisfying one soft constraint may make it impossible to satisfy another one. Therefore, the order by which the soft constraints are posted to the constraint engine becomes important. In the case that some soft constraints are mutually inconsistent to each other, constraints posted the last (or those closer to the leaf nodes of the search tree) are backtracked into before those posted earlier (or those closer to the root node of the search tree).

The problem at hand is to decide an order of importance among the soft constraints. There are three areas in which such ordering can be based on: (1) among each planning rule as a whole, (2) among the days for those constraints which affect a series of days, and (3) among the nurses. The first order is a user-defined parameter, which reflects the priority of different planning rules. The second and third ordering is generated randomly. This ensures that no human factor is involved in the decision process, in which fairness is an important criterion.

Rules 9 and 11 are difficult to model as constraints in the constraint-solving engine. We devise a sequential algorithm to allocate the shift for the following reasons. First, allocation of N shifts is subject to a number of difficult constraints. Second, the process involves many history records such as the date of the last N shift for each nurse, date of last E shift for each nurse, and so on. Third, there are always cases that some nurses cannot take N shift even after they have been assigned to do so because of nurse preference shift requests.

The algorithm in Figure 3 is based on the idea that the earlier the date of last shift (type of which can be N or E) of a nurse is, the sooner the shift should be allocated to the nurse. For example, suppose nurse X and Y have their last N shifts on November 4 and November 6 respectively. Further suppose that there are two N shifts to be allocated on November 12 and 13. It is normal to assign the N shift on the 12th to nurse X and the 13th to nurse Y. However, suppose nurse X has pre-set shift SD on November 12, causing X unable to take N shift on that day. An obvious answer is to exchange the shift with nurse Y so that nurse Y takes N shift on the 12th and nurse X takes N shift on the 13th.

Suppose there is further complication that nurse Y has also pre-set shift on the 12th. Then there must be another nurse replacing the shift. The main goal of our algorithm is to specify how to choose such nurse to replace the shift.

```
procedure allocate-shift(nurse[], shift-type, shift-num)
  discard unavailable nurses for shift-type for the week
  sort nurse[] according to the date of last shift-type
  i = size(nurse[]) − shift-num
  if i < 0
    copy the first i nurses in nurse[] to its end
  endif
  i = 0
  day = 1
  while shift-num > 0
     cur-nurse = nurse[i]
     if cur-nurse is unavailable for shift-type on day
       swap cur-nurse with one in filled-nurse[]
       if not successful
         swap cur-nurse with nurse[j] (j > i)
         if not successful
           swap cur-nurse with a discarded nurse
           if not successful
             replace cur-nurse with one in filled-nurse[]
             if not successful
               output "No shift can be allocated"
             endif
           endif
         endif
       endif
     endif
     filled shift-type for cur-nurse on day
     insert cur-nurse into filled-nurse[]
     increment(i)
     increment(day)
     decrement(shift-num)
  endwhile
endprocedure
```

Figure 3. N and E shifts Allocation Algorithm

4.5. Solution Generation

Execution of the nurse rostering engine is divided into four steps. First, all shift pre-assignment requests are collected. Second, the N and E shifts allocation algorithm is invoked to assign E and N shifts. Third, the rest of the constraints (hard or soft) induced by the planning rules (imperative or preference) are collected and told to the constraint engine. Fourth, a standard variable labeling algorithm is used to search for solutions. There are two things to note regarding variable labeling. First, since the aggregate model contains the slot model and the shift model as sub-models, labeling variables of either model will instantiate also variables in the other model. We can choose to label either set of variables. Second, we employ the simple smallestdomain-first variable-ordering heuristics and the smallest-first value-ordering heuristics in the variable labeling process. This corresponds to the a simple application of the first-fail principle [3].

4.6. Implementation

We implement the rostering system on personal computer (PC), which is chosen over workstation for PC's wide availability. Our implementation 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 [4]. Class constraints [4] of ILOG SOLVER facilitate our object-oriented design and greatly reduce our coding effort.

Our real-life benchmark data contains 27 nurses and 11 shift types. The rostering system produces weekly roster sheet[3]. The implementation of the slot model generates 3884 constraints, out of which 266 are imperative, during execution. Executing the shift model 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 aggregate model consists of, in addition to the constraints from the slot and the shift models, 2079 channeling constraints.

5. Benchmarking Results

We have tested the system based on real-life data from the hospital. The tests differ in terms of the number of shift pre-assignments and soft constraints posted. There are several ways to generate solutions from the system. First, we can search for solutions purely based on the slot model. That means solution generations amount to posting all the constraints in the slot model and instantiating the slot variables only. Alternatively, we can use the shift model, and generate rosters by instantiating the shift variables only. In both cases, the system performs satisfactorily (about 10 seconds for building up variables and the constraint network, 10 seconds to get the solutions) in the problem instances where no or few pre-set shifts are present and only one to two soft constraints are posted. When there are more pre-set shifts or more soft constraints, the system fails to generate solutions within 10 hours, irrespective of the choice of models.

The other way to get rosters from the system is from the aggregate models. As mentioned before, forming an aggregate model from sub-models involves channeling constraints which connect the variables in one model to variables in another model[4]. About 20 seconds are needed to build up the variables and constraints in this model, and again solutions can be obtained within 10 seconds. Most importantly, many cases which are unsolvable in either the slot or shift model can be solved efficiently in this model within 10 seconds. In the following, we present the timing result of the three models for a particularly difficult problem instance.

Figure 4 shows the pre-set 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 pre-set shifts. Among them, 5 nurses request the O shift on Sunday, which is most wanted by other nurses as well. 2 nurses request VL, which extends across the entire week. Last but not least, one nurse is assigned SD from Monday to Friday and requests for O on Sunday. As a result, 47 out of 189 slots are pre-set before the roster can be generated. Some of the pre-set 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 pre-set requests, we vary the combination of preference rules posted and show the performance of the three models in Figure 5. Column one of each row specifies the combination of preference rules imposed. Columns two and three contain the timings of the slot and shift models 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. Column four shows the timing result of the aggregate model by labeling only variables in the slot model. Alternatively, we can label only variables in the shift model and yield the results in column five. Experimental results confirm that the aggregate model exhibits significant speedup over either the slot or the shift model.

During our implementation, the generated rosters are sent to the hospital duty officer for evaluation. He compares the rosters with his manually generated counterparts, and returns comments to improve the quality of the solutions. We follow his advise and improve upon the system. This iterative process lasts for about three months, and now the system is ready for pilot run.

6. Concluding Remarks

The contribution of this project is three-fold. We have successfully completed the construction of a real-life nurse

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

[4]The aggregate model introduces more memory overhead. However, with the advent of cheaper and more massive hardware, we do not foresee this as a major obstacle

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		

Figure 4. A Difficult Rostering Problem

rostering system, which consists of a constraint-solving engine and an ergonomic user-interface. It demonstrates the potential of constraint technology in the computerization of hospital management. Instead of consuming time in generating the rosters, hospital officers can spare more time and effort on patients and other medical duties. Other areas of hospital management, such as nurse duty assignment, patient-ward allocation, should also benefit from constraint technology.

In the development process of the core engine, we observe that techniques in solving classical CSP's alone are often insufficient to cope with real-life problems. In particular, soft constraint is not part of a standard CSP, and yet it is an important element in our problem. Our adoption is to model soft constraints as choices in the solution search tree, but this leads to a considerably large search space, thus increasing solution search time. We alleviate this problem by applying a novel technique, redundant modeling. Benchmarking results confirm substantial improvement in solution generation efficiency.

A future direction of development is to design alternative methods for implementing soft constraints in the constraint-solving engine where no notion of soft constraints is provided. These methods include designing variable- and value-ordering heuristics and optimizing objective functions which reflect the quality of solution. Hierarchical and stochastic constraint methods can also be investigated for their suitability to cope with soft constraints.

Van Emden *et al* [15] suggest that system users do not usually know exactly what they want, and yet afterwards they do know what they do not want. This observation holds true in our implementation experience in this project. When we first meet with the duty planner, he gives us a set of planning rules. In subsequent meetings, he often discovers that he either misses or mistakes some details, thus leading to changes to the old planning rules or even creation of new planning rules. Yet we are able to cope with these frequent changes and modify our implementation accordingly. We

Soft Rules	Slot Model	Shift Model	Aggregate Model	
			Slot Vars	Shift Vars
—	short	short	short	short
7+16	long	long	fast	1 min
7+8	long	long	1 min	long
7+8+16	long	long	fast	long
7+8+16+13	long	long	fast	long
7+16+13	long	long	fast	long
7+13	long	long	long	long
7	8 min	5 min	1 min	2 min
8	1 min	long	fast	long
16	long	fast	fast	fast
12	1 min	fast	fast	fast
13	long	fast	long	fast
11	long	fast	fast	fast
15	long	long	fast	long

Figure 5. Benchmark Results

attribute our flexibility to the expressiveness of constraint programming. Changing the problem specifications, in most cases, simply implies modifying the domains of the variables and expression of the associated constraints. Such high level of flexibility is difficult to achieve using mathematical programming or problem-specific algorithms.

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. Mr. Tam is also always available to answer our queries. Constructive comments from the anonymous referees help improve the quality of the final version of the paper.

References

[1] P. C. Bell, G. Hay, and Y. Liang. A visual interactive decision support system for workforce (nurse) scheduling. *INFOR*, 24(2):134–145, 1986.

[2] B. Cheng, J. Lee, and J. Wu. Speeding up constraint propagation by redundant modeling. (to appear) Second International Conference on Principles and Practice of Constraint Programming, August 1996.

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

[4] ILOG. *ILOG Solver Reference Manual, Version 3.0*. ILOG, 1995.

[5] M. M. Kostreva and K. S. B. Jennings. Nurse scheduling on a microcomputer. *Computers Operations Research*, 18(8):731–739, 1991.

[6] J. M. Lazaro and P. Aristondo. Using SOLVER for nurse scheduling. In *Proceedings of ILOG SOLVER & ILOG SCHEDULE First international users' conference*, July 1995.

[7] A. Mackworth. Consistency in networks of relations. *AI Journal*, 8(1):99–118, 1977.

[8] J. D. Megeath. Successful hospital personnel scheduling. *Interfaces*, 8(2):55–59, 1978.

[9] H. E. Miller. Nurse scheduling using mathematical programming. *Operations Research*, 24(8):857–870, 1976.

[10] I. Ozkarahan and J. E. Bailey. Goal programming model subsystem of a flexible nurse scheduling support system. *IIE Transactions*, 20(3):306–316, 1988.

[11] C. L. Pape. Using object-oriented constraint programming tools to implement flexible 'easy-to-use' scheduling systems. In *Proceedings of the NSF Workshop on Intelligent, Dynamic Scheduling for Manufacturing*, January 1993.

[12] J.-F. Puget. A C++ implementation of CLP. In *Proceedings of SPICIS 94*, November 1994.

[13] S. U. Randhawa and D. Sitompul. A heuristic-based computerized nurse scheduling system. *Computers Operations Research*, 20(8):837–844, 1993.

[14] S. P. Siferd. Workforce staffing and scheduling: Hospital nursing specific models. *European Journal of Operational Research*, 20:233–246, 1992.

[15] M. van Emden, M. Ohki, and A. Takeuchi. Spreadsheets with incremental queries as a user interface for logic programming. *New Generation Computing*, 4:287–304, 1986.

[16] D. M. Warner. Scheduling nursing personnel according to nursing preference: A mathematical programming approach. *Operations Research*, 24(8):842–856, 1976.

[17] G. Weil, K. Heus, P. Francois, and M. Poujade. Constraint programming for nurse scheduling. *IEEE Engineering in Medicine and Biology*, July 1995.