A Hybrid AI Approach to Staff Scheduling

Graham Winstanley
School of Computing & Mathematical Sciences
University of Brighton
g.winstanley@bton.ac.uk

Abstract

Assigning staff to specific duties according to their contract, qualifications, skills, etc. within a working environment characterised by multi-disciplinarity and statutory regulations is problematic. This paper discusses an approach to nurse rostering, using a strategy of distributing the computational effort required in the scheduling process. The technique involves a hybrid approach that devolves responsibility for different aspects of the problem. In the pre-processing stage, the staff to be rostered are treated as semi-autonomous agents, each equipped with heuristics to guide their initial assignment. Compilation of individual rosters is followed by a scheduling phase in which a constraint solving agent applies constraint logic programming (CLP) techniques in the generation of 'acceptable' rosters.

1. Introduction

The generation of work allocation schedules can generally be summarised as the assignment of staff to particular time slots in order to satisfy given criteria. Time slots may be specified shifts, or they could be temporal intervals with dynamic start and end times. The criteria can be simple or complex, sometimes involving legal, corporate and safety issues in addition to heuristics that govern individuals and ever-changing institutional policies. The rostering of nurses is widely accepted as an important and challenging intellectual problem that belongs to the class NP-complete. Many studies have been undertaken in the area, using techniques such as mathematical programming [1] heuristic methods [2] and constraint satisfaction techniques [3], [4], [5], [6]. There are even commercially available systems, whose authors claim that their products are capable of wide application [7]. In all these cases, the authors are in agreement that the combinatorial search problem is the most difficult to solve. Heuristic methods such as that proposed in [2] attempt to circumvent the search problem by identifying specific rules that could be applied to progressively fill in a roster. However, although such methods are efficient in initial search space pruning, i.e. pre-processing, they are generally not complete.

Mathematical programming continues to be used for this class of problem and does provide completeness, but with the always-present overhead of an enormous search space. Constraint logic programming (CLP) has been widely used in recent times, but almost always within a hybrid architecture that commonly includes a high degree of user interaction and heuristic pre-processing [3]. CLP offers what appears to be an ideal vehicle for the solution of scheduling problems such as nurse rostering. The high-level nature of logic programming is augmented by the seamless integration of one or more constraint solvers, thus allowing the programmer the comfort of modelling the problem and its constraints declaratively. With this approach, the programming steps are, in outline: model the problem, declare the constraints, and apply a specific search algorithm to find a satisfactory or optimised solution. Modelling includes the definition of variables, data and data structures, domains of variables, etc. Declaring constraints on the problem variables is catered for by the syntax rules of the constraint solver and can be unary, binary or n-ary. Search is the final step and is always necessary with problems of realistic scale. It proceeds by systematically instantiating variables from their domains and testing all constraints on that variable and any variable related to it (via constraints) [8], [9].

Unfortunately, CLP, or in fact any of the above-mentioned techniques alone, is not capable of solving the nurse scheduling problem, even when esoteric methods such as double-modelling are employed [4]. In our study there were a total of 18 nurses and 11 possible working shifts. For a one month scheduling period this amounts to a search space of $11^{(18*28)}$, however this is reduced slightly by reducing the domain size. Most successful systems appear to be based on the application of several methods of representation and reasoning, and this is the approach adopted in our work. Another unfortunate fact is that no two nurse rostering situations appear to be identical. This means that one solution, based commonly on the identification of symmetries and applicable heuristics becomes so specific to that particular situation, that its generality is lost. Commercial systems allow the user to predefine the 'business rules' in a kind of 'system tuning' process, but this places the emphasis very much on the knowledge and skill of the person(s) assigned to optimise the system for each application. Our approach has been to partition the problem into its identifiable components and perform a solution in a number of phases, including heuristic methods and CLP.

Results from the use of the system on a hospital medical ward within the UK NHS have been successful. Additionally, the application of the Staff Work Allocation Tool (SWAT) on the ward has proved to be an invaluable vehicle for gaining a better understanding of the interplay of constraints at various levels, i.e. from individual to institutional, and national levels.

2. Distributing the problem

Initial research into the process of rostering identified the naturally distributed nature of the problem. In many ways it resembled industrial domains characterised by team working, and particularly interdisciplinary collaboration. Complexity in such domains has been shown to be solvable using a multi-agent approach in which individuals are represented by semi-autonomous software agents that are able to solve problems local to themselves, but also to co-operate together to achieve predefined global goal(s) [10]. In producing a nurse roster, one person is usually given the (unenviable) task of allocating staff to shifts, i.e. time slots, for each day in a scheduling window of commonly one month. In doing that job, he/ she must be aware of the skill and/ or qualification profile of the staff, and be fully cognisant of all requirements and constraints which control how the finished roster should look. In some large hospitals, such rosters may be produced centrally, but experiences have shown that most commonly the task is performed by a staff member on the ward that is being rostered, and in our case that process was characterised by a great deal of local negotiation. Each staff member knew their qualifications and nursing grade. Each one had some preference for working patterns and each individual had the opportunity to request certain shifts on certain days or weeks. Over a period of several months, patterns seemed to evolve that serve to make the scheduler's job (apparently) easy. However, in many cases, partly due to requests that 'upset' the preset patterns, annual leave, sickness, etc., the task becomes very difficult indeed.

The agent-based metaphor adopted in our work assumes the following: Each staff member could be given a copy of the blank ward roster. Individual rosters could then be produced, based on peoples' knowledge of their own situation as it relates to the roster. Relationships may exist between individuals that result in constraints between them, but essentially negotiation would be assumed to have taken place before individual schedules are produced, i.e. the definition of preferred patterns of work would be accomplished prior to scheduling. The staff member assigned to the task of ward scheduling would then collate individual schedules into a ward roster. Given the unlikely situation that no deviations from the agreed preferred shift patterns had taken place, through requests say, the resulting ward roster should be, by definition, satisfactory without further effort. In the far more usual case, the roster production task would involve minor 'shuffles' of shifts. An optimal solution to this would be to minimise such perturbations, and an approach to dealing with this method of final ward rostering is discussed later in this paper.

The problem has the following characteristics:

1. Staffing comprises 'trained' staff and 'untrained' staff. Trained staff are professionally qualified and registered. Untrained staff are commonly referred to as auxiliary, or nurse assistants. To cater for the inevitable over-constrained case, the hospital has the benefit of a 'nursing bank.'

2. Requests are generally honoured. The Ward Manager is responsible for approval, but once this has been given, requests are treated as hard constraints.

3. On each and every day, there should be a predefined staff establishment and shifts are standardised as: Early full, Late full, Late half and Long day

4. There should be no 3 consecutive long days. If a staff member has worked two long days already, the next shift must be a day off.

5. In our approach, days off are used generically for any day not spent on the ward, i.e. including annual leave, sickness, study leave, etc. in order to reduce the number of shifts and therefore the search space [11].

6. Bank staff should be employed only when necessary. In practice, established staff members are commonly asked to work extra shifts. This poses some interesting problems in automating the process, but our solution has focused on the incorporation of bank shifts to address the over-constrained case.

Agents are defined in our system for each member of nursing staff, with a common architecture. Information pertinent to each agent is stored, and specific pre-processing is carried out at the individual agent level before the roster management phase.

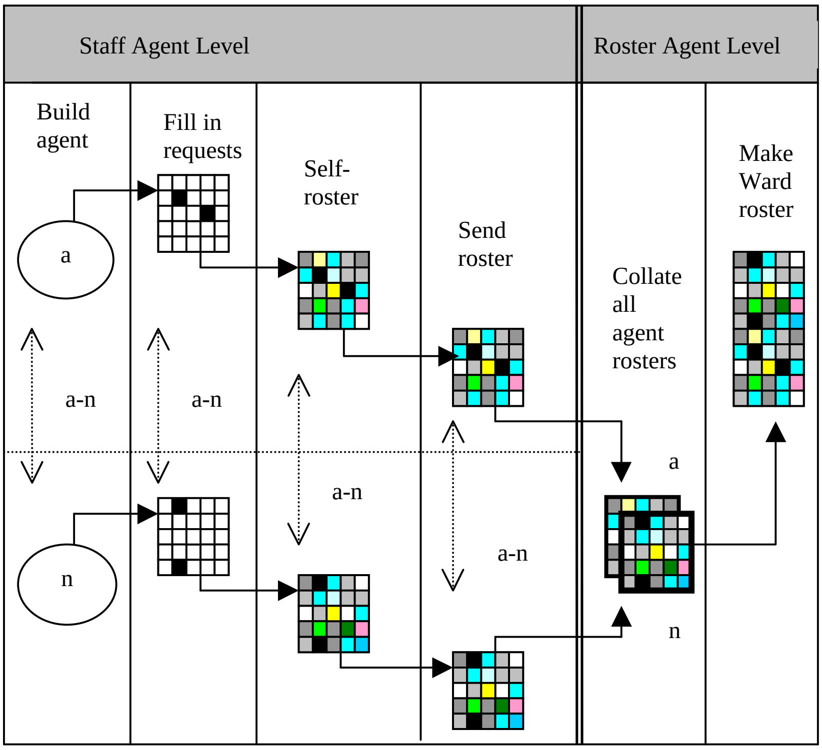

Figure 1. Sequential phases involved in producing a ward staff roster

The various phases involved in the process are shown diagrammatically in Figure 1. With reference to this figure, our solution to the rostering problem involves a number of phases, based on the following assumptions:

- Each agent should be capable of 'filling in' their own part of a ward schedule and taking part in a collective and possibly negotiated agreement.

- Preferred working patterns represent a ‘starting point’ for scheduling. In an ideal situation, these shift patterns, when collated together, would provide a working solution for the ward establishment for the day/ week / month. In practice, changes would almost always be required.
- Requests for specific duties, leave, etc. are evaluated at the local, i.e. individual agent, level. Once these requests are accepted, they must be honoured.
- Production of the final ward roster is the responsibility of the Roster Manager Agent. This agent is controlled by a number of higher-level constraints pertinent to the ward, the institution and beyond.

3. Architectural issues

The architecture of the SWAT System is based on the object-oriented knowledge system Kappa-PC , loosely coupled with the ECL^{i}PS^e Constraint Logic Programming System. Each staff member is represented as an agent with the following components:

- A ‘core’ profile component. This component holds data on staff name, grade, qualifications, contracted hours of work, status and preferred working shift patterns.
- A request component that doubles up to contain the initial individual schedule. This component holds data on any requests that have been processed and agreed by the Request Manager Agent.
- A personal constraint component that holds specific types of constraint, e.g. ‘X must work with Y.’ The data in this component is accessed when defining new preferred work patterns and when processing requests.
- A communications component which currently relays object messages. There is no specific agent communication language or protocol at work.

An object called ‘Nurse’ is defined, with subclasses ‘Trained’, ‘Untrained’ and ‘Bank’, and three methods: ‘BuildAgent’, ‘RemoveSelf’ and ‘SelfSchedule.’ BuildAgent takes data obtained from the user interface and creates and appropriately names the above agent components. Figure 2 shows the agent architecture as an object hierarchy for 2 nurses in each of the 3 categories. In this figure, solid lines signify class/ subclass relationships, and dotted lines signify instance relationships.

In addition to the nurse agents, there are also three manager agents defined: ‘RequestManager’, ‘RotaManager’ and ‘InterfaceManager’, which are equipped with methods to deal with individual requests for days off, shifts, etc., invoking the initial scheduling system and controlling the interface to the user and the Roster Manager Agent, which exists and operates outside the object-oriented environment described in this section.

Figure 3 shows the main interface to the system. Using this interface, it is possible to view, add, delete and modify staff details, view requests and initial rosters for each staff member. It has facilities to view, add, delete or modify requests, and it is from here that initial and final ward scheduling is invoked.

With reference to Figure 3, and Figure 1, in the process of producing individual rotas for each agent, the following stages have taken place:

1. Requests are made for days selected from the (main) month displayed on the interface. Rules exist to validate requests and ensure that, once accepted, they persist through subsequent reasoning phases.

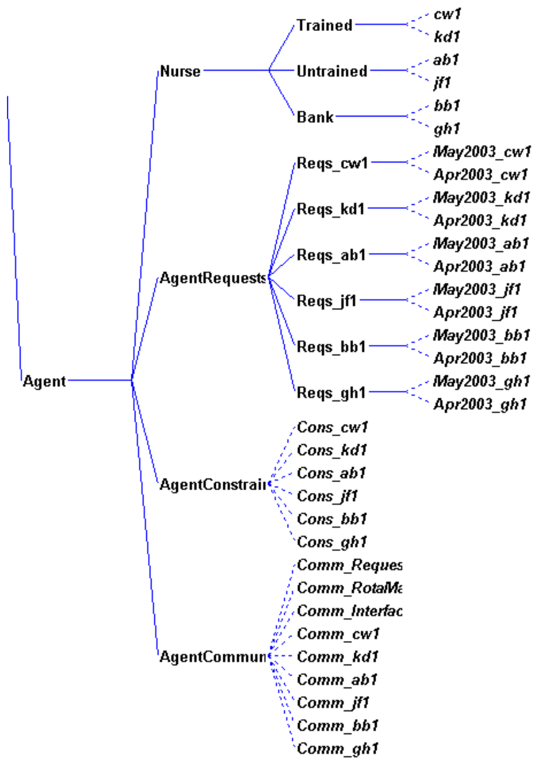

Figure 2. The object hierarchy for the nurse agent

2. A forward-chaining inference process invokes 'self-scheduling.' For each agent defined within the system and labelled as 'active', the process involves four phases:

   Phase 1: Scans the month on a day-to-day basis and assigns requests with a numerical value to signify a hard constraint. Requests are

	processed first and are always honoured after being accepted by the backward-chaining inference process.
Phase 2:	Scans the month on a day-to-day basis and checks for requests being in conflict with preferred assignments. If such conflicts are detected, this phase records the details.
Phase 3:	This phase takes the data recorded in Phase 2 and re-allocates shifts that were preferred, but have now changed due to requests. Modifications to the shift pattern for the week in question are minimised.
Phase 4:	This is only active when preferred shifts are de-selected. This phase involves a large and complex rule set that effectively labels each day with a shift according to predefined soft constraints.

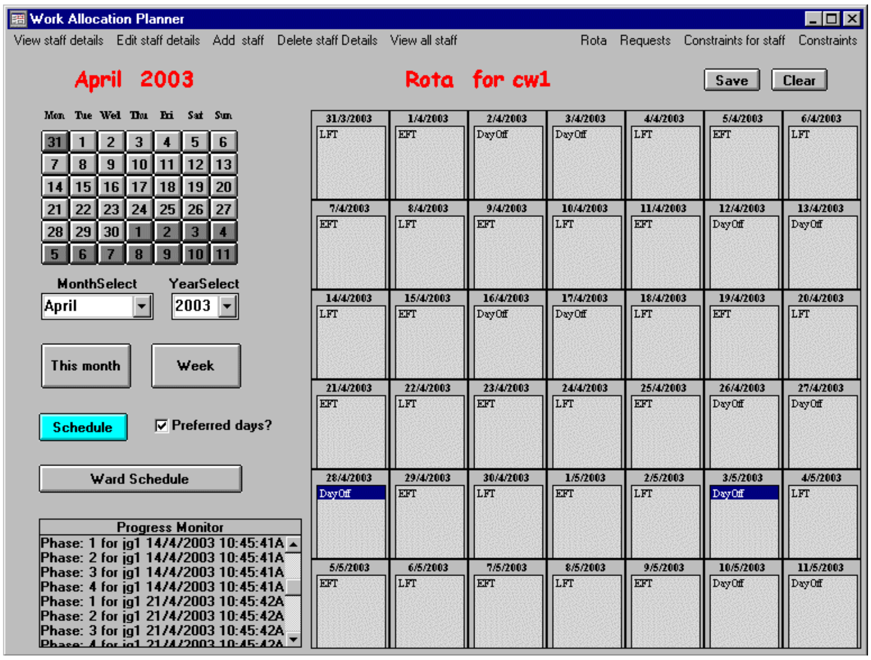

Figure 3. The system's main interface

Figure 3 shows that the staff member 'cw1' had requested the day off on Monday 28 April and Saturday 3 May, and that the 'normal' preferred shifts for that week had been slightly modified. Agent schedules are stored as ASCII files, and the Roster Manager Agent is subsequently invoked on that data. The first file contains data on the (short) names for each agent, their contracted time and their possible shifts

The second data file holds the actual initial schedules in the Prolog-style format shown below:

```
data([(1 ,apr ,2003 ,eft ,6 ,cw1),
(2 ,apr ,2003 ,dof ,6 ,cw1),
:
:
(28 ,apr ,2003 ,dof ,9 ,cw1) ]).
```

This indicates that the staff member ‘cw1’ has an early full trained shift on 1 April, 2003, and it comes about as a preferred shift. This agent has requested a day off on 28 April.

4. The Roster Manager Agent

Corporate constraints are crucial in any ward roster. They govern the number of staff at various levels, they dictate the patterns of work, and they control staff working hours. Every hospital ward has a staff establishment, i.e. the number and spread of personnel for each day in order to comply with legal standards and the guidelines developed and agreed at the corporate level. In the SWAT System, these constraints are modelled and applied at the ward level, compiling individual agents’ rosters and applying its constraints to each day and to all staff. These constraints include:

1. Number of trained and untrained staff per day. The constraints have a lower and upper limit.
2. Staff should not be scheduled for a work pattern that leads to their contracted hours of work being exceeded. This is quite a powerful constraint since the hours of work ‘value’ can be manipulated to cater for overtime in a controlled way. It is also possible to facilitate the inclusion of bank staff during the many occasions when a roster is simply impossible with the available staff (holidays, sickness, etc.).
3. Certain staff working patterns are ‘preferred.’

The roster manager is responsible for collating individual rosters for one month, and applying specific constraints. Its task is to create viable rosters according to constraints propagated from the individual agent phase, and to corporate constraints at the ward level. However, minimal changes should be made to the initial roster because this represents the ‘ideal’ situation according to the wishes and preferences of individual staff. In other words, repairs should be made only when necessary, and those repairs should cause minimal disruption to the initial roster.

The ECL^{i}PS^{e} Constraint Logic Programming platform [12] was used in the production of the final roster subsequent to the agent self-scheduling and roster compilation phases. This platform facilitates the conceptual modelling of the nurse rostering problem in Prolog style, and readily provides a mapping between that model and the program required to solve the problem, i.e. the design model. It has the great advantage of being a constraint solver at a high level, closely integrated with the Prolog programming language, and it is equipped with a number of constraint solving methods within its libraries. The facility to define a high-level specification of the problem at the conceptual level and to similarly define constraints at the design level, leads to the ability to rapidly produce such models and to experiment with them. In combinatorial problem domains, this method of design-by-experimentation is tractable.

Modelling in ECL^iPS^e follows closely the principles adopted in the agent-based component of the SWAT System. Each agent produces and communicates a matrix of shifts, with data filled in for each day of a one-month staffing period. This data is in the form of a list with the following structure:

{day, month, year, shift_type, constraint_hardness, staff}

Other data is communicated, such as the range of possible shifts for each agent, i.e. the domain for each agent. Status information is also available, such as trained or untrained.

This is incorporated into the system as a number of supporting structures, and two key arrays are defined:

1. A two-dimensional array with axes for days (1-28) and staff (1-18). Each cell of this array is constrained to be instantiated by values from the domain of values for the relevant agent. The cells are then filled in from data provided by the agent-based component of the SWAT System, and become 'tentative values' for later constraint solving.
2. An identical array, but this time having only (initially) uninstantiated variables. This array provides the structure for the variables and the constraint solving process. As constraint-solving proceeds, values are chosen and evaluated according to the constraints on them and between other values.

In this structure, there are i staff and j days. Each cell of the array is a variable Vi,j that must be instantiated to a shift type, including non-shifts, i.e. days off, n total staff and t total days to schedule.

The first, and most important constraint at the ward level relates to the staff establishment on the ward for each day. This constraint, in ECL^iPS^e syntax is defined as:

```
shift_occurrences(et,Day_list,N1), N1#>=2, N1#<=3      % et=early trained
shift_occurrences(eut,Day_list,N2), N2#>=2, N1#<=3     % eut=early untrained
shift_occurrences(lt,Day_list,N3), N3#>=2, N1#<=3      % lt=late trained
shift_occurrences(lut,Day_list,N4), N4#>=2, N1#<=3     % lut=late untrained
```

Where the 'shift_occurrences' constraint is based on the built-in ECL^iPS^e 'occurrences' constraint. The 'et' argument signifies early trained, etc., Day_list is a list of variables made available for each day, i.e. {Vi,j} for $n \geq i \geq 1$ and for j = the day in question. The SWAT System loops through all $t \geq j \geq 1$, applying these constraints, which generally say that "for each day, there should be at least 2 trained on an early and late shift, but at most 3, and there should be at least 2 untrained on an early and late shift, but at most 3." The specific shift types that correspond to these abstract shift types are defined within the shift_occurrences constraint itself. These constraints are also equipped with annotation that controls propagation and specifies the ECL^{i}PS^e repair library, but this has been ommitted for reasons of clarity.

It is important to constrain the system to give staff only those combinations of shift types that sum to their contracted hours. This constraint is quite easy to model in ECL^{i}PS^e, as shown below:

```
add_them(Z31,Z32,Z33,Z34,Z35,Z36,Z37,Z22):-
        (Z31 + Z32 + Z33 + Z34 + Z35 + Z36 + Z37) #=Z22.
```

where Z31 to Z37 are variables that hold the time value for each shift assigned, and Z22 is the contracted hours value for the staff in question. It declares that the summation of assigned shifts is constrained to be equal to the contracted work time. The system loops through $n \geq i \geq 1$ and for each week within the one-month period.

The assignment of shifts according to constraints on their patterns is catered for in the system as below. For each staff i, and for $t \geq j \geq 1$, R=the hardness value of Vi,j. A value of 9 signifies that the shift should be assigned exactly as it appears in the equivalent cell in the tentative value array (Init_shift_array). A value of 6 means that the value can be changed, so long as the new value $\in$ {domain of i}. P is Vi,j and P2 is the tentative value of Vi,j.

```
(R == 9) -> (P = P2)    % if a hard constraint, then impose the tentative value
```

The constraint ‘no more than two consecutive long days’ is defined as:

```
Q is Shift_array[I,J-1],                      % Q is yesterday’s shift
Q2 is Shift_array[I,J-2],                     % Q2 is the day before’s
  ((Q#=ldt) #/\ (Q2#=ldt)) #=> (P#=dof)       % #/\ is the ‘and’ constraint
  ((Q#=lda) #/\ (Q2#=lda)) #=> (P#=dof)       % #=> is the ‘then’ constraint
```

which says that ‘if the last two days were long days, for trained and untrained, then the next day must be a day off. Other constraints have been experimented with to cater for informal patterns, such as early shifts before days off and late shifts after days off. However, in practice these were seen to frequently over-constrain the problem and are commonly violated in practice. These constraints have a similar structure to the one shown above and remain in the system as optional features.

The strategy described below is designed to reduce the search complexity by addressing the problem in four one-week chunks. Complexity and tractability issues are also important in the algorithm chosen for search itself. To deal with the common problem of no roster being possible with staff numbers available, the algorithm first tries for a solution with all available staff. If no roster is possible, a bank shift is added and search begins again. This continues until all possible bank hours have been utilised, and if no roster can be found under these circumstances, the system terminates, reporting a failure.

The algorithm controlling the Roster Manager is, in outline:

Read in data for each agent into a number of non-logical variable structures
Create arrays for initial shift values, status, staff domains, staff work hours
Apply domains to initial and final shift arrays
Until a consistent set of value assignments has emerge, i.e. one that satisfies all constraints
For each week in the one-month scheduling period
  Set the initial shift array to its tentative values (from the previous phase)
  Apply constraints: staff numbers and patterns, hours worked
  Search: The systematic search for values consistent with the applied constraints
 If no set of value assignments can be made then
  If there are bank staff hours left then
   Increase bank staff hours by one shift
   Restart search
  Else
   Terminate with failure
 Else
  Terminate with a satisfactory roster

5. The Roster Manager search strategy

The strategy that underpins the SWAT System assumes an initial tentative assignment of all staff to shifts according to their preferences and personal constraints. Therefore, a constructive labelling approach that iteratively searches for values would be unnecessarily complex. Our methodology involves the 'repair' of tentative values in an effort to satisfy the constraints defined at the roster manager level. At one extreme, the tentative values satisfy all constraints, and are accepted as given. At the other extreme, all tentative values may have to be changed and search complexity is at its theoretical maximum for the nature and scale of the problem at hand. In the case of manual nurse rostering, this repair strategy is common, and in practice has never been seen to reach the worst case. Various move-based strategies have been proposed to deal with this type of problem, notably [13], [14]. We have chosen to use the iterative improvement/ backtracking hybrid algorithm [14], which has been termed 'weak-commitment search.' In weak-commitment search, constraints are defined on variables and the tentative values of variables. In the case of constraint violations occurring, search for variable assignments is guided by a heuristic that chooses to instantiate a variable with a value from its domain that minimises constraint violations with the tentative values of unlabelled variables. This heuristic is due to [13] and is called 'min-conflict.' When a situation arises where no value can be found that satisfies all of these constraints, this combination of assignments is remembered as a 'no-good' constraint and the combination will not be tried again. Search is then restarted with the current value assignments as the new tentative values. This approach to search has important implications for automation of the current nurse rostering problem. It assumes (requires) a tentative solution, which is tested against all constraints. In the case of a failure at this stage, a value is chosen from the domain of a variable that causes the minimum conflict with the tentative values of the as yet unlabelled variables. A partial solution is therefore built in a sensible way by 'repairing' only those tentative assignments that are seen to be problematic. If no consistent value assignment can be made for any variable, then instead of backtracking to the last variable assignment, weak-commitment search abandons the whole path, recognising and storing the fact that this was a 'no-good' assignment set, and therefore should not be tried again. It is this weak commitment to the current branch in the search space that gives the algorithm its name. Once a dead end has been reached, the algorithm stores the bad assignment set as a no-good constraint, and starts the whole process again from scratch. However, the current value assignment set becomes the new set of tentative values. The assumption here is that this represents a 'better' solution than the previous one and does not throw away the manipulative work of the previous stage of the search.

6. Experiences with the system

The SWAT System has been tested on a hospital medical ward as detailed in this paper and has performed at least as well as manual scheduling according to reports from those responsible for rostering on that ward. Experience indicates that there are many aspects to the production of a workable roster that includes: the ability to negotiate 'on the fly' with staff immediately before roster publication, and commonly afterwards. Although the removal of this kind of 'bartering' is generally considered a good thing by those involved in the rostering process, the automated schedules were regularly criticised for being too rigid. However, this appears to have cultural roots, since once the system had produced a sequence of such rosters, criticisms became less frequent. The fact that the final rosters reflected the initially chosen preferred shifts allowed the Ward Manager to refine these patterns over time in such a way that harsh changes were mostly minimised.

Evaluation of the system occurred in two ways. The first measure of success involved user satisfaction, from the point of view of the scheduler and of the staff being rostered. The second involved the time taken to produce a roster. On average, using preferred shifts, the initial heuristic phase was completed in approximately 3 minutes. The final rosters were created as a secondary phase in a similar time. Using weak-commitment search, the number of restarts (equivalent to backtracks in backtracking search) was commonly one or two, but never more than 5. The ward currently uses a paper diary to record requests, and this became the primary source of data entry for phase one. However, considerable simplification and standardisation of requests and request types were required. Future versions will constrain requests to specific types. Initial individual rosters are accessed in the Phase 1 system, and the final ward roster is output as an ASCII file for publication in a spreadsheet format. At present there is no feedback from Phase 2 to Phase 1 other than the ability to manually criticise the two roster types and make changes to preferred shift patterns. This mixed initiative approach has many advantages, none the least in staff appreciation, but there is obvious scope for automation and adaptation.

7. Conclusions

The system has been in use for several months and has been successful in producing timely and accurate ward staff rosters. The hybrid approach taken in the development of the SWAT System, along with its agent-based architecture, has proven to be natural and intuitive for this class of problem. Devolving initial scheduling to software agents representing individual staff was a core development decision and has proved to be an effective solution to the problem. Agents can be dynamically created, activated, modified in and deleted from the system, and each one reasons locally, but provides sufficient information to the roster manager agent to facilitate the creation of ward rosters based on CLP techniques. The ECL^iPS^e system allowed us to model the problem and facilitate the dynamic nature of information passed from Phase 1. Constraints were relatively easy to define and test in this system, and the search strategy used resulted in a reduced search space. It should be noted that combinatorial space and time complexity remains and can cause severe problems in the worst case, but experience has indicated that this is unlikely, especially with the incorporation of 'bank staff' to cater for the over constrained situations.

There remains much research to undertake. Our heuristic Phase 1 could easily be replaced by a distributed CLP system, but at the moment some degree of flexibility would be lost at the pre-processing stage where information can be acquired, stored, manipulated and generally reasoned with quite efficiently at the heuristic level. A partial constraint satisfaction solution, in which a range of 'soft constraints could be defined and allowed to be violated in a controlled way in the quest for a 'good' solution would also be useful. However, our experiences and experiments with such approaches has led us to believe that, in the case of nurse rostering, these extra overheads have a marginal effect on the final product.

8. References

1 Miller, H.E., Pierskalla, W.P., Rath G.J. Nurse scheduling using matematical programming, In Operations Research, Vol. 24, No.8, pp857-870, 1976.

2 Isken, M.W., Hancock W.M. A heuristic approach to nurse scheduling in hospital units with non-stationary, urgent demand and a fixed staff size, Journal of the Society for Health Systems, Vol.2, No.2, 1991.

3 Abdennadher, S., Schlenker, H. INTERDIP – An Interactive Constraint Based Nurse Scheduler, Proceedings of the 1st Int. Conf. on The Practical Application of Constraint Technologies and Logic Programming, PACLP99, London, 1999.

4 Cheng, B.M.W., Lee, J.H.M., Wu J.C.K. A Nurse Rostering System Using Constraint Programming and Redundant Modelling, IEEE Transactions on Information Technology in Medicine, Vol.1, pp44-54, 1997.

5 Weigel, R., Faltings, V.B., Choueiry B.Y. Context in Discrete Constraint Satisfaction Problems, 12th European Conference on AI (ECAI96), pp205-209, Budapest, Hungary, 1996.

6 Scott, S., Simpson, R. Case Bases Incorporating Scheduling Constraint Dimensions: Experiences in Nurse Scheduling, In Advances in Case-Based Reasoning (EWCBR98), Springer Verlag Lecture Notes in AI, 1998.

7 Shibutzit. www.shibutzit.com Accessed April 2002

8 Kumar, V. Algorithms for Constraint Satisfaction Problems: A Survey, AI Magazine Vol.13, No.1, pp32-44, 1992.

9 Jaffar, J., maher M. Constraint Logic Programming: A Survey, Journal of Logic Programming, Vol.19, No.20, pp503-582, 1994.

10 Nunez, J., Winstanley, G., Griffiths R. N. A Reluctance-Based Cost Distribution Strategy for Multi-Agent Planning, The International Journal of Applied Intelligence, Special Issue on *Intelligent Adaptive Agents*, Vol.9 pp39-55. Kluwer Academic Publishers, The Netherlands, 1998.

11 Freuder, E.C. Eliminating interchangeable values in constraint satisfaction problems, Proceedings of the 9th National Cnference on Artificial Intelligence (AAAI-91), pp227-233, 1991.

12 Wallace, M., Novello, S., Schimpf, J. ECLiPSe : A Platform for Constraint Logic Programming, IC-Parc, 1997. http://www.icparc.ic.ac.uk/eclipse/reports/eclipse/eclipse.html, Accessed July 2001.

13 Minton, S., Johnston, M. D., Philips A. B., Laird P. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, Artificial Intelligence, Vol. 58., pp 161-205, 1992.

14 Yokoo, M. Weak-commitment Search for Solving Constraint Satisfaction Problems, Proceedings of 12th Nat. Conf. On Artificial Intelligence, WA, USA, pp 313-318, 1994.