S27-63
137255

N93-18688

p/5

Combining Constraint Satisfaction and Local Improvement Algorithms to Construct Anaesthetists' Rotas

Barbara M. Smith

Division of Artificial Intelligence
School of Computer Studies
University of Leeds
Leeds LS2 9JT, U.K.

Sean Bennett

Department of Anaesthetics
The General Infirmary at Leeds
Great George Street
Leeds LS1 3EX, U.K.

Abstract

A system is described which has been built to compile weekly rotas for the anaesthetists in a large hospital. The rota compilation problem is an optimization problem (the number of tasks which cannot be assigned to an anaesthetist must be minimized) and has been formulated as a constraint satisfaction problem.

The forward checking algorithm is used to find a feasible rota, but because of the size of the problem, it cannot find an optimal (or even a good enough) solution in an acceptable time. Instead, an algorithm has been devised which makes local improvements to a feasible solution. The algorithm makes use of the constraints as expressed in the CSP to ensure that feasibility is maintained, and produces very good rotas which are being used by the hospital involved in the project.

It is argued that formulation as a constraint satisfaction problem may be a good approach to solving discrete optimization problems, even if the resulting CSP is too large to be solved exactly in an acceptable time. A CSP algorithm may be able to produce a feasible solution which can then be improved, giving a good, if not provably optimal, solution.

The Rostering Problem

Leeds General Infirmary (L.G.I.) is a large teaching hospital in the centre of Leeds. The anaesthetics department consists of 19 consultant anaesthetists and 24 other full-time anaesthetists in more junior grades, who are referred to collectively as junior anaesthetists. The junior grades are primarily training grades, and part of the junior anaesthetists' training is to work alongside a consultant anaesthetist. However, in the U.K., junior anaesthetists also do some work on their own. At the L.G.I., there is a set of operating lists, referred to as junior lists, which are always covered by junior anaesthetists working on their own. Junior anaesthetists may also be required to cover consultant lists on their own if the consultant is away. The consultants work the same pattern of operating lists every week, but a weekly rota is required for the juniors, showing what each will be doing in each of ten weekly sessions (Monday to Friday, a.m. and p.m.).

There are three grades of junior anaesthetist: Senior Registrar (SR), Registrar and Senior House Officer (SHO), in descending order of seniority. The SRs and half the Registrars are assigned for a month or more at a time to a training block, which is a specialty such as paediatrics, in order to improve their skills in that area. Most of the SRs work to a fixed timetable in their own specialty for most or all of the week, assisting a consultant. The Registrars who are on a training block should also work with the consultant in their training specialty for much of the week, although they do not have a fixed training timetable. The remaining Registrars are assigned to General Duties, and are available to cover junior lists, stand in for absent consultants and so on, for most of the week, as are the training block Registrars when not involved in training. The SHOs are not assigned to a particular specialty, but are doing general training in the specialties not covered by the training blocks; the least experienced SHOs should spend most of their time accompanying a consultant, while those with more experience can do some of the junior lists on their own, or stand in for an absent consultant.

One of the SRs is assigned to the 'General Duties/Admin' block, and the administrative part of this is to compile the weekly rota. The Admin SR spends half a day a week compiling the rota for the following week. Since each SR spends a maximum of six months on this block, the Admin SR is only becoming expert at compiling the rota by the time that the next person takes over. The job therefore takes much more time than it would if the same person did it all the time; it is also difficult to ensure consistency. On the other hand, the person compiling the rota needs to be an experienced anaesthetist, in order to know what specialties different people can cope with on their own, and so on. The Admin SR is also responsible for making any adjustments to the rota after it has been compiled, for instance if someone is ill, and needs to be able to judge whether a particular operating list will be relatively straightforward, or requires someone with considerable experience in the specialty. Hence it is not appropriate to entrust the compilation of the rota to a clerk, but it was felt that a system which could produce the initial rota automatically, under the control of the Admin SR, would be of great benefit. It would also allow more strategic questions to be explored, for instance, how many anaesthetists of each grade are required to cover the operating workload.

The rota varies from week to week, partly because of the on-call rota. This is compiled separately, for a month at a time, and shows for each night of the week, and the weekend, five junior anaesthetists who are on call to deal with emergency work, for instance in obstetrics or the Intensive Care Unit. For Registrars and SHOs, being on call at night governs what they do on the immediately previous and following days. The rota also varies because of staff absences, which result in changes to the work that needs to be allocated in the week. If an SR is away, in many cases the work that he or she would have done has to be assigned to someone else, preferably to the Registrar working in the same specialty, if there is one. If a consultant is absent, sometimes no action need be taken, for instance if an SR would normally assist the consultant, and can take responsibility for the list instead. Often, however, a junior anaesthetist who is capable of doing the list alone must be found. When junior anaesthetists are absent, the work to be done has to be shared amongst fewer people; in some weeks the level of absences means that several operating lists have to be cancelled. Compiling the rota therefore means solving a different problem each week: the work to be done varies ¿From week to week, as do the personnel available to do it.

If the opportunities for Registrar and SHO training are included, it is not possible to compile a weekly rota which covers all the work, and the Admin SR tries to strike a balance between covering as many operating lists as possible and allowing adequate training. The first priority, however, is to cover those lists where the consultant is absent; the junior lists can, if necessary, be left uncovered, in which case the list is cancelled, and it is not essential that juniors should be assigned to all the training lists available.

Compiling the Rota

The first step in compiling the rota for a given week (whether manually or by computer) is to record the planned absences of each junior anaesthetist and their predetermined assignments, i.e. those due to regular commitments or to the on-call rota. This gives a partly completed rota, the gaps showing where the juniors are still available to do the remaining work. The Admin SR then needs to know which other operating lists need to be covered and which training lists are available in that week, given the planned absences of both consultants and juniors. This gives a set of tasks to be done in each session of the week, together with a set of people available to do them. In addition, some anaesthetists must be assigned an half day off during the week: normally, an afternoon off is taken following a night on call, but if an anaesthetist is not on call during the week, an afternoon off has still to be assigned. A half day for the compilation of the next rota must also be set aside for the Admin SR.

The rota compilation system extracts the set of tasks to be done, and the junior anaesthetists available, from its basic information about the department, which does not change from week to week, and from data on absences and the on-call rota, which does need to be input each week. The departmental data includes, for each consultant operating list, the action to be taken if the consultant is away: various strategies are available, for instance, to assign a specific junior anaesthetist if they are available, and failing that one of the different grades of junior, listed in order of preference.

Compiling the rota then consists of assigning an anaesthetist to each task, taking into account the requirements of the different tasks, e.g. some operating lists require a particular grade of anaesthetist, some training lists are only appropriate for the anaesthetist training in that specialty, and so on. At the same time, the additional afternoons off must be assigned. The rota must be optimized, in the sense that the number of tasks left unassigned must be minimized, while a satisfactory balance is kept between training and covering the junior lists.

The number of tasks to be done varies from week to week, but is normally about 90-100, and the number of anaesthetists who can do each task averages about 5.5. The number of anaesthetists who need to be given a half day off is about 4 or 5. The size of the problem can be reduced if we recognize that some of the training lists, i.e. the general training lists which are principally for SHOs, rather than the specialized training lists attached to the training blocks, are of much lower priority than other tasks. Acceptable rotas can be compiled by assigning the other tasks first, and then fitting the general training lists into the remaining gaps. This reduces rota compilation to two separate problems, the second of which is trivial. The first problem then has about 75 tasks to be assigned.

Constraint Satisfaction Problems

The constraint satisfaction problem has been discussed extensively in the Artificial Intelligence literature [see references]; it can be used as a formulation of many problems arising in OR. In a constraint satisfaction problem there are a number of variables, each of which has a discrete set of possible values (its domain). There are also a number of constraint relations, specifying which values are mutually compatible for various subsets of the variables: for instance, the assignment of an anaesthetist to a task is incompatible with the assignment of the same anaesthetist to another task in the same session. A solution to the constraint satisfaction problem is an assignment of values to the variables which satisfies the constraints.

Although the definition of the CSP does not distinguish between solutions, so that all assignments which satisfy the constraints are equally acceptable, it is possible to represent optimization problems as CSPs. The objective is represented as an additional constraint, which changes each time a new solution is found. For instance, in a minimization problem, the constraint is that the value of the objective must be less that its value in the best solution found so far (or, initially, less than some very large number). This ensures that each solution is better than the previous one, and when all the solutions to the CSP have been found, the last one will be optimal. A similar scheme for representing discrete optimization problems as CSPs is described by van Hentenryck [3].

In general, constraint satisfaction problems are NP-complete, so that although several algorithms exist for solving them ([2], [4]), they are not guaranteed to find a solution in a reasonable time unless the problem is small or has special structure. However, in many cases there is a good chance of finding a feasible assignment quite quickly. Optimization problems, on the other hand, will almost certainly suffer from the exponential worst-case performance, since the search cannot be terminated when the first feasible solution is found. Despite this difficulty, it may still be possible use a constraint satisfaction formulation as a basis for finding good solutions to optimization problems, as demonstrated below.

Nadel [4] surveys the available algorithms for the CSP, and compares their performance on some standard problems. One of the best algorithms in these experiments is the forward checking algorithm, described by Haralick and Elliott [2], and this algorithm is used by the rota compilation system.

The Rota Compilation Problem as a CSP

As mentioned earlier, the first stage in compiling the rota is to record the predetermined assignments and the planned absences for the week. The CSP formulation will only be concerned with the problem of assigning the remaining tasks to those anaesthetists who are still available after this first stage.

The variables of the CSP are used to represent the tasks to be assigned in the given week, and the domain of each variable is the set of anaesthetists who can do that task. In addition, there is a small number of variables which represent a half day off for an individual anaesthetist. The domain of such a variable is the list of sessions in which the anaesthetist could take a half day off.

The domain of each task variable is arranged in priority order, with the best choice of junior anaesthetist for the task appearing first. The forward checking algorithm selects values from the domain in the order in which they appear, and hence the anaesthetist appearing first in the list is the one most likely to be assigned, if available. Although ordering the domains is not guaranteed to give the overall best allocation of anaesthetists to tasks, it does in practice give acceptable results.

In order to express the relative priorities of the different types of task, they are divided into three categories: essential, preference and optional. The essential tasks are those arising from consultant absences: an anaesthetist must be assigned to each of these in order to achieve a feasible solution. (It is extremely unlikely that a situation could arise in practice where consultant absences could not be covered.)

The preference tasks correspond to the junior lists and the Registrar accompanied lists, i.e. those training lists which allow a Registrar to accompany a consultant anaesthetist in their assigned specialty. To allow the algorithm to leave the preference tasks uncovered if necessary, an extra value, NONE, is added as the final element in the domain of each of the corresponding variables. When this variable is considered by the algorithm, this value can be selected, if all the anaesthetists who could do this task have been assigned to something else.

It has been found that a satisfactory balance between covering the junior lists and assigning the Registrars to training lists in their own specialty can be achieved by covering as many of the preference tasks as possible, i.e. the number of preference tasks assigned the value NONE should be minimized. This can be done by using an additional constraint to represent this objective, as described in section 3.

The final category is the optional tasks: these are the training lists for the SHOs, in which they accompany a consultant. These also have the value NONE as the last element of their domain. SHOs can be assigned to these tasks if there is nothing of higher priority which they could do instead; to reflect this, the optional tasks are assigned only after a satisfactory assignment of the essential and preference tasks has been found. The current state of the rota is then fixed and the optional tasks are assigned to those junior anaesthetists who have not so far been allocated to do anything in that session.

The constraints of the CSP firstly arise from the fact that an anaesthetist cannot do two things at once, so cannot be assigned to two task variables in the same session, or to have a half day off at the same time as doing a task. These constraints may be thought of as general rostering constraints; similar constraints expressing the fact that no-one can be assigned to do two tasks at the same time will occur in any rota compilation problem. The anaesthetists' system also has a constraint representing the objective, as already described.

In addition, there are other constraints reflecting particular rostering rules used at the L.G.I., which have in fact changed several times during the course of the project. Currently, for instance, there is a rule that Registrars who are on a training block can be taken off training, and assigned to a junior list instead, at most once during the week. Constraints of this kind are likely to vary from hospital to hospital and, as experience at the L.G.I. has shown, to change over time. The system has therefore been designed in such a way that constraints are easy to express.

Improving a Feasible Solution

Having set up the variables and their domains, the forward checking algorithm is used to find an assignment of the essential and preference tasks and the half-day variables. Very little backtracking is required to find a feasible assignment, because most variables do not represent essential tasks and so can if necessary be assigned the value NONE, which, at this stage, does not conflict with any other assignment. The algorithm therefore finds a first feasible solution very quickly. However, because of the size of the problem, finding the optimum solution would take a very long time. Often, finding any improvement to the first solution takes far longer than would be acceptable.

It is possible that improvements in the way that the forward checking algorithm is used might achieve a sufficient increase in speed to allow an optimal solution to be found. For instance, there are variable and value ordering heuristics, such as those discussed by Nudel [5] which can be expected to give significant improvements in appropriate cases. Value ordering heuristics cannot be used in this case because the original ordering of the domains must be preserved, and the variables with smallest domains cannot be assigned first, as is commonly advised, because they do not represent tasks which are hard to assign, but rather the Registrar training lists, which should not be given higher priority than other tasks. It is still conceivable that variable ordering rules based on problem knowledge could be developed. However, rather than pursuing this possibility, we have used the forward checking algorithm only to produce a feasible solution, and looked for ways of improving such a solution. This approach produces good results very quickly, and it seems unlikely that an improved forward checking algorithm would be able to do any better.

In order to improve on the best solution that the forward checking algorithm can find quickly, an algorithm has been devised that considers each uncovered task in turn and looks for reassignments of related tasks which will allow it to be covered. This local improvement algorithm was developed through examining feasible but non-optimal rotas, and looking for reassignments that would improve them.

Suppose that there is an uncovered task that we want to try to find an assignment for. This is a variable which has been assigned the value NONE. All the anaesthetists in the variable's original domain must have been assigned to do something else in this session (otherwise the assignment of NONE would not have been made) but it may be possible to free one of these anaesthetists by reassigning the task that they are currently assigned to (a *swap*), or by moving a half day off from this session to another session (a *move*).

The following example (adapted from an actual rota) shows the kind of swaps within a session that can be made in order to improve the solution.

Variable	Original Domain	Assigned
ORTHO-TRAUMA-THU-AM	(R-4 R-6 R-5 SHO-1 SHO-2 NONE)	R-4
CW-II-THU-AM	(R-4 R-6 R-5 SHO-1 SHO-2 NONE)	R-6
OBS-THU-AM	(R-5 R-4 R-6 NONE)	R-5
GARDNER-THU-AM	(R-5 NONE)	NONE
PSU-I/A-THU-AM	(R-4 R-6 R-5 NONE)	NONE

The variables are shown in the order in which the forward checking algorithm considers them, so that the value assigned is the first remaining value in the domain. (Values assigned to other variables representing tasks in this session have been omitted.) The two uncovered operating lists in this Thursday morning session (GARDNER and PSU-I/A) can be covered by making use of SHO-1 and SHO-2 who are so far unassigned in this session. The simpler swap is to assign the ORTHO-TRAUMA list to SHO-1, thus allowing R-4 to do the PSU-I/A list. Covering the GARDNER list entails a chain of two exchanges: SHO-2 takes the CW-II list, R-6 takes the OBS list, and R-5 can then do the GARDNER list.

A simple example of a move is to move an anaesthetist's half day off from a session where there is an uncovered task that this anaesthetist could do to another session where they have not been assigned to do anything. More complicated changes involve a swap, of the kind illustrated above, combined with a move. This is done if moving a half day off would allow an uncovered task to be done by the anaesthetist concerned, and the swap has to be done to free the anaesthetist in the session that the half day off is being moved to.

The local improvement algorithm considers each uncovered task in turn in the current solution, and for each anaesthetist in the original domain of the corresponding variable, each of the above changes is tried, starting with the simpler changes, until a change which will allow the task to be covered is found, or the variable's domain is exhausted. This procedure ensures that the first value in the domain which can be assigned to the task is found, thus observing the preference ordering of the values.

In all cases, potential changes to the current solution are checked against the constraints, so that even when new constraints are introduced (e.g. an upper limit on the number of junior lists a Registrar on a training block can do in a week, as mentioned above), the algorithm still produces a feasible solution.

The local improvement algorithm works through the list of uncovered tasks once, and then presents the resulting solution as the best that it can achieve. The combination of swaps and moves seems to be adequate to produce an optimal rota; so far, we have not been able to see any further scope for reducing the number of uncovered tasks in the rotas produced, except by relaxing the constraints.

Producing the Rota

At this point, the rota will have several gaps, where an anaesthetist has not been assigned to do anything. The final stage in constructing the rota is to assign the optional lists to fill these gaps. The resulting rota is then printed out, with a note of any remaining uncovered junior lists.

The Admin SR may still wish to make changes to the rota before it is issued. This is partly because there may be places in the rota where an anaesthetist has not been found anything to do; since the workload varies so much from week to week, there are often sessions where there are fewer tasks than available anaesthetists, as well as sessions in the same week where sessions have to be left unassigned. The Admin SR can assign spare anaesthetists to give additional assistance at operating lists which have already been covered. Occasionally, when there are outstanding unassigned tasks, the Admin SR may be able to relax the constraints in order to allow them to be covered. Even when the system does not produce immediately usable rotas, the remaining tidying-up takes only a few minutes: the difficult part of the job has been done.

Alternative Approaches

Dhar and Ranganathan [1] describe a similar problem to rota compilation (that of assigning teaching faculty to courses) and compare an integer programming formulation to an expert system. In their expert system, production rules are used to express both problem solving knowledge and constraint knowledge. In the rota compilation problem, however, expert problem solving knowledge is not easily available. The Admin SR changes every few months, so that there is not usually sufficient time to develop any great expertise and there is little opportunity to pass on experience from one incumbent to the next; each person therefore evolves their own method of rota compilation, based largely on trial and error. It seemed best, therefore, to use an algorithmic approach to constructing the rota and to use the successive Admin SRs only as a source of constraint knowledge.

There is scope, however, for making more use of problem solving knowledge in rota compilation. For instance, at present there is no attempt to identify the session which will be most difficult to cover and to assign the tasks in that session first. Hitherto, this has not been important because there has been little interaction between the different sessions; the constraints are for the most part between tasks in the same session. If the interaction between sessions increased, then it could become important to use this kind of problem-solving knowledge, by using it to direct the order in which the forward checking algorithm considers variables.

Results and Conclusions

The rota compilation system has been developed in Common LISP on a Sun 3/160; it is now also running on a PC. It can produce a weekly rota within 30 minutes, including entering the required data, compared with the half day allocated to compiling the rota manually. The system has been producing good quality rotas for the L.G.I. for over a year, and has coped with changes in the rota compilation rules. We are currently improving the user interface so that the system can be used by hospital staff. In future, we intend to investigate similar problems in other hospitals and to extend the system to deal with them.

Apart from the fact that the system saves the Admin SR several hours work each week, with less risk that a task will be forgotten, another benefit is that it can be used to evaluate different policies, reflected in different sets of constraints. A series of rotas which would result from the different policies can be produced and compared, using real data on absences, etc., from past weeks. Hitherto, there has been no way of evaluating the effects of proposed changes in policy.

A common approach in Operational Research to optimization problems which cannot be solved exactly is to find (somehow) a feasible solution and then to look for local improvements which will hopefully produce an acceptable solution. Incorporating the two stages, of finding a feasible solution and then improving it, into the constraint satisfaction framework has a number of benefits. First, constraint satisfaction seems a natural way of formulating many discrete optimization problems; there is a close correspondence between the variables and values of the CSP and problem entities. In OR approaches, on the other hand, especially those based on mathematical programming formulations, there may be a significant translation gap between the original problem and its formulation. Secondly, since there are already CSP algorithms, a means of finding a feasible solution is readily available: it is not necessary to write a special-purpose algorithm.

Finally, the local improvement algorithm can make use of the constraints, as expressed in the CSP formulation, to ensure that any changes maintain feasibility. This has been demonstrated in the rota compilation system, when a new constraint has been introduced. Adding a constraint to the CSP requires only a few lines of LISP; the local improvement algorithm needs no modification at all, since it merely checks any potential changes against the new constraint. Hence, building the local improvement algorithm within the CSP framework gives a very flexible and easily modified system, which would be hard to achieve otherwise. Although the system described here is very specialized, the general approach of finding a feasible solution and then improving it, all within the CSP framework, is one that might be applicable to many optimisation problems in scheduling.

References

[1] V. Dhar and N. Ranganathan (1990) Integer Programming vs. Expert Systems: An Experimental Comparison, *Communications of the ACM* **33**, 323-336.

[2] R.M. Haralick and G.L. Elliott (1980) Increasing Tree Search Efficiency for Constraint Satisfaction Problems, *Artificial Intelligence*, **14**, 263-313.

[3] P. van Hentenryck (1989) *Constraint Satisfaction in Logic Programming*, MIT Press.

[4] B.A. Nadel (1989) Constraint Satisfaction Algorithms, *Comput. Intell.* **5**, 188-224.

[5] B.A. Nudel (1983) Consistent Labeling Problems and their Algorithms: Expected Complexities and Theory-Based Heuristics, *Artificial Intelligence* **21**, 135-178.