A Hospital Placements Allocation Problem

S.N. Cresswell and T.L. McCluskey
School of Computing and Engineering
The University of Huddersfield, Huddersfield HD1 3DH, UK

Abstract

We describe a novel allocation problem which has arisen in selecting training placements for students on a university course in Operating Department Practice (ODP). We formalise the problem's constraints, and develop programmed, CLP and ILP solutions, and compare them. The final solution using CLP techniques has been embedded in a user interface and is being evaluated by members of our ODP department.

1 Introduction

In this paper, we describe and solve resource allocation problems which have arisen in selecting training placements for students on a university course in Operating Department Practice (ODP). The ODP course is a two-year course in which a number (c. 60) of students experience a number (c. 7) of hospital placements in each year. Each of the placements is either a surgery or anaesthetic placement in a specific medical speciality.

The course organisers are considering a change to the management of placements in which the management of allocation is no longer delegated to hospital trusts, but managed centrally by the course organisers. This will allow students to have a greater variety of experience by visiting more than one hospital, and enables bottlenecks in the availability of placements in particular specialities at particular hospitals to be circumvented, thereby giving the potential of increasing the overall capacity of the course. The problem is inherently complex - there are a number of constraints to be taken into account, such as capacities of hospitals in various areas of speciality, ensuring each student has a full diet of specialities, allowing students to attend hospitals that are close to their home, etc.

The problem can be solved in an ad hoc way by a human planner, taking days or weeks of effort. However, the quality of the solution may not be high, and shifting capacities may call for re-work of the schedules at short notice. For example, a new student may have to be integrated into an existing schedule. Further, the need arises for estimating the actual number of students that can be placed, given a fixed set-up of hospitals and specialities.

Given the inherent combinatorics of the problem, we developed several heuristic solution methods with the goal in mind of eventually creating a customised placements allocation tool for the clients. As the requirements shifted we eventually settled on a CLP solution, encased in a user friendly interface. In the next section we describe the problem in more depth, then go on to formulate the constraints for the initial requirements we were given. In the following section we analyse the problem and develop three solution methods. These are then compared with respect to the goals of the system. Finally, we detail a set of extra requirements and outline work leading to a CLP solution of this more complex problem.

2 The placements problem: version 1

2.1 Informal account of the problem

Initially, we will assume allocation is for the *first year* of the 2 year course only. The resources of the problem are within a set of hospitals (c.12) distributed between towns in a region. Leeds has three 3 hospitals, and Bradford, Huddersfield, Halifax, Dewsbury, Wakefield, Pontefract, Keighley and Harrogate all have one. Each hospital has specialities with a maximum student resource capacity (if we allow student capacity=0 then we can assume all hospitals can have the potential of offering every speciality). The specialities currently considered are General Surgery, Gynaecology, Urology and Orthopaedics.

Information is needed on each student's preferences for hospitals to be attended. This is based on the idea that students will commute to hospitals, and so there will be a subset of the hospitals that they can commute to/from. In version 1, we assume this subset of hospitals is specified for each student, and is unordered.

Each speciality is broken down into two sections – anaesthetic mode and surgery mode. Students must attend the two modes of a speciality in consecutive placements. Furthermore, a student should always alternate between anaesthetic and surgery on cosecutive placements, even when moving to a different speciality. We choose to pair the placements so that students remain in the same speciality and hospital for the pair of placements, and only change between surgery and anaesthetic. This means that we only deal with 3 double-sized time slots. We label the paired placements according to their *phase* – i.e. whether a double timeslot comprises anaesthetic then surgery, or surgery then anaesthetic. The alternation requirement is satisfied if a student's allocations are all of the same phase.

To summarise, the constraints are:

Reachability of hospitals Placements must be within reasonable commuting distance from home location of student. Each student chooses a subset of available hospitals which are within commuting distance.

Non-repetition Each of 6 placements is in either anaesthetic or surgery in one of the specialities.

Capacity There are a number of hospitals, and all have a limited capacity (usually 0-2) for the number of placement students that can be accepted in each speciality.

Alternation A student should not have: 2 consecutive placements of anaesthetic, or 2 consecutive placements of surgery

The goal of the problem is:

- *to produce an allocation of students to placements which meets all the constraints.*

The availability of placements is the main factor limiting the expansion of the university course. A secondary goal is therefore:

- *to determine how many more students can be accommodated under the central placements system.*

A small part of an example schedule is shown in figure 1, and was generated by one of the tools we developed.

Student	P2	P3	P4	P5	P6	P7
1	Wakefield gen anaesthetic	Wakefield gen surgery	Dewsbury gynae anaesthetic	Dewsbury gynae surgery	Wakefield urology anaesthetic	Wakefield urology surgery
2	Huddersfield gen surgery	Huddersfield gen anaesthetic	Calderdale ortho surgery	Calderdale ortho anaesthetic	Huddersfield urology surgery	Huddersfield urology anaesthetic
3	Calderdale gen surgery	Calderdale gen anaesthetic	Bradford ortho surgery	Bradford ortho anaesthetic	Calderdale gynae surgery	Calderdale gynae anaesthetic
⋮						

Figure 1: Example Schedule

2.2 Formulation of the Problem

Throughout we will use the same set of names to denote the important parameters of the problem:

h	hospital
st	student
sp	speciality
t	timeslot
ph	phase

Next we introduce some pre-defined integer functions for returning the capacity of a hospital in a speciality – i.e. the maximum number of student places available during any timeslot, and the set of hospitals that is within a reasonable travelling time for a given student. Given the input of a student and a time period, the output value is a tuple denoting the hospital, speciality and phase of the speciality (i.e. whether it is anathetic or surgery, or surgery and then anaesthetic:

Input data

$cap(h, sp)$	integer capacity of hospital h in speciality sp
$reachable(st)$	set of hospitals reachable by student st

Output data

$alloc(st, t)$	allocation of student st at time period t, allocation is tuple $\langle h, sp, ph \rangle$

Given this notation we can now proceed to formalise the constraints discussed in the section above:

Capacity The number of students allocated to a particular hospital, speciality and phase is within available capacity. The capacity of a hospital/speciality is assumed to be constant throughout the time period. To capture this, we express the constraint by fixing every value of h, sp, t and ph, and checking that the size of each set of students is less than the capacity:

$$\forall h.\forall sp.\forall ph.\forall t.\, |st : alloc(st,t) = \langle h, sp, ph\rangle| \leq cap(h,sp)$$

Reachability of hospitals. student can only be allocated to reachable hospitals. To formulate this constraint, we fix the student and time-slot, and check if the hospital in the solution is also in the set of reachable hospitals for that student.

$$\forall st.\forall t.\exists h. \left[\begin{array}{c} \exists sp.\exists ph. alloc(st,t) = \langle h, sp, ph\rangle \\ \wedge \\ h \in reachable(st) \end{array}\right]$$

Non-repetition . Each student's allocated speciality is unique – it is not repeated in any other timeslot

$$\forall st.\, |\{sp : \exists t.\exists h.\exists ph. alloc(st,t) = \langle h, sp, ph\rangle\}| = 3$$

Alternation For each student, there is one phase used in all timeslots.

$$\forall st.\exists ph.\forall t.\exists h.\exists sp. alloc(st,t) = \langle h, sp, ph\rangle$$

3 Solution Techniques

3.1 Introduction

The existing practice of allocation of students to hospitals had been managed using pen and paper. Decisions as to which students go to which hospital had been made by individual health trusts, rather than being the subject of central (student-centred) planning. Part of the reason for automation was to assist in a change to such a student-centred arrangement. The University Staff in the ODP department were the primary source of knowledge of the problem formulation. Past records of student details, hospital capacities, and successful schedules were available indirectly, to be used to verify any candidate solutions.

As with any real problem of this nature, the requirements of the solution, and some of the domain knowledge, is destined to change over time and particularly during the time of the project. The domain experts will learn about the capabilities of automated solutions, and discover more about the constraints of the problem during the project. Hence, it is essential in this kind of project to develop potential solution methods that are flexible and adaptable to requirements change. This turns out not to be easy, as a small change in problem characteristics could render the problem intractable.

From a scheduling point of view, hospitals can be thought of as *resources*, a student's year is a *job* and the specialities are the *activities*. Students use up available and reachable 'resources' in order to fashion a course of activities for themselves. However, some of the other parallels with scheduling are missing: jobs all run in parallel, all have the same format, and all have the same 'makespan'; where we have a generalisation of scheduling is that activities are not fixed - any subset of the specialities (activities) that fill the year are possible.

The problem can be tackled from a timetabling point of view. In this case, we are developing circa 60 timetables, as each student's timetable could be (but is not necessarily) unique.

Further, as mentioned above, the set of specialities that a student could enjoy is not fixed a priori - only the set of possible specialities is fixed. Thus what is timetabled can be chosen to suit the allocation problem.

Hence, the problem does not seem to fit neatly into a traditional scheduling or timetabling area. Having analysed the problem as atypical, we decided to investigate several solution techniques in parallel. This flexibility was needed in the face of likely changing requirements. We developed a solution using a declarative programming language, programming in constraints, as well as trying various constraints programming and scheduling packages.

3.2 Programmed heuristic solution

The first step was to attempt to develop a programmed solution. This is useful in order to:

- enable a greater understanding of the problem
- provide a baseline comparison with packaged solvers

However, the programmed solution is likely to be inferior with respect to maintenance and the ability to adapt to changing requirements.

The programmed solution was designed to allocate one student's timetable, then move on to the next student. Thus a depth first solution was taken in terms of committing and allocating all of a student's time slots before considering the next.

The constraints of Capacity, Reachability, Non-repetition and Alternation were built into the timetable generation.

For each student st:
    For each slot t:
        generate a tuple (h, sp, ph) from
            the set of reachable hospitals,
            the set of specialities not yet allocated,
        and taking into account the phase of the student.

When generation fails during search, this means that at some point there is no non-repeated speciality in a reachable hospital available in the student's phase. Backtracking undoes allocations of a previous time slot, and then of previous students, then re-allocates options if possible, and retries the current student with the new options.

We used Sicstus Prolog to implement this method. Using depth-first search, without heuristics the search would immediately hang. Crucial to the success of this simple approach was to use a domain-specific heuristic determining solution order for students. The heuristic is as follows: *The number of hospitals each student can reach determines the order each student is considered - the most tightly constrained students are allocated 'resources' first.*

3.3 Constraint logic programming

Constraint logic programming (CLP) [1] allows the creation of a declarative model expressing the variables and constraints of the problem. Here we make use of a finite domain constraint solver, which associates with each variable a domain of possible values. As choices are made, the constraints are used to progressively eliminate values from the domains of related variables.

Languages incorporating finite domain constraint solvers, such as ECLiPSe [1], GNU Prolog [2], and Oz [2] offer a range of useful built-in constraints with which to model the problem. Below, we describe how the model is constructed using standard built-in constraints of a finite domain solver.

Solver variables:

- $alloc(st, t)$ - each tuple $\langle h, sp, ph \rangle$ is represented by an integer value.
- $spec(st, t)$ - the speciality in tuple value of $alloc(st, t)$
- $phase(st)$ - the phase of all allocations to student st

Constraint types:

Capacity - For each hospital, speciality, phase and time, the *atmost* constraint is used to constrain the number of instances of the tuple $\langle h, sp, ph \rangle$ in student allocations.

Reachability - a priori pruning of domains of $alloc(st, t)$.

Non-repetition - The $spec(st, t)$ are related to the corresponding $alloc(st, t)$ by means of an *element* constraint. We ensure that for each student, the values taken by $spec(st, t)$ are distinct in each time slot by means of an *alldifferent* constraint.

Alternation - $phase(st)$ is linked with indexes of compatible tuples in all time slots.

The above model was implemented using an FD module in the Oz programming language [3]. The creation of the model is separated from the process of searching for a solution. We initially tried a built-in search strategy, which is a fail-first heuristic [3][4]. This orders the constraint variables such that values are first chosen for the variables with the smallest remaining domains. The idea behind this is that the most constrained part of the problem is tackled first.

Note that the heuristic can be seen as similar to the programmed heuristic solution, except that the ordering is done dynamically rather than fixed in advance, and the number of available $\langle h, sp, ph \rangle$ tuples is considered instead of hospitals only.

3.4 ILP model

Integer linear programming can be used where all constraints can be expressed as linear (in)equalities [5]. The model presented below does not deal with all of the constraints, but ignores phase constraints. The ILP model has some characteristics that are different from the previous approaches. It is most appropriate for optimising a linear objective function, which we do not have. However, it has been useful to run the solver on the relaxed problem because it is able to detect infeasibility, and hence we were able to get an upper bound on the maximum number of students. To do this, it is not really necessary to restrict the solver to integer solutions, as infeasibiity of the continuous relaxation of the problem, which is computationally cheaper to test, implies infeasibility of the integer problem.

[1] http://www.icparc.ic.ac.uk/eclipse/

[2] http://pauillac.inria.fr/ diaz/gnu-prolog/

[3] Implementation note: the CLP solution was built using Mozart/Oz 1.3.1, including graphical interface on the final system. Experiments with linear programming solvers were done with lp_solve integrated with Oz via XRI and LP modules.

We use a matrix of variables taking 0-1 integer values, $alloc(st, h, sp, t)$. A constraint is required to ensure that there is exactly one allocation for each student in each timeslot:

$$\forall st. \forall t. \sum_{h} \sum_{sp} alloc(st, h, sp, t) = 1$$

Capacity

$$\forall h. \forall sp. \forall t \sum_{st} alloc(st, h, sp, t) =< cap(h, sp)$$

Non-repetition

$$\forall st. \forall sp \sum_{h} \sum_{t} alloc(st, h, sp, t) =< 1$$

Reachability

$$\forall st. \forall h \, [h \notin reachable(st) \rightarrow \forall sp. \forall t. alloc(st, h, sp, t) = 0]$$

Phase constraints are not naturally encoded as linear inequations, so not all constraints are handled here.

4 Experimental Evaluation

Our experiments were aimed at evaluating the techniques with respect to the primary and secondary goals described above. Two tests was carried out using two versions of real student data from past years. This was obtained and collated from our ODP department and from various hospitals in the area. In order to establish the maximum possible number of students, extra students were added to the data until solutions could no longer be found. In the case of the Programmed and CLP code, this means that search continued for more than 10 minutes. In the case of the ILP program, the problem was reported to be infeasible.

	$\leq max$		$\geq max$
	Programmed	CLP	ILP (relaxed problem)
prob1	66	69	73
prob2	71	72	75

Figure 2: Maximum number of students allocated by each algorithm.

The results are shown in the Table 2. The solutions generated by the programmed and CLP methods for 56 real student records were acceptable to our clients. We then added more student records until the method failed to give a solution - in the first test this was 67 for Programmed, and 70 for CLP. We were unsuccessful in modelling all of the constraints in the ILP model, hence the solution to the relaxed problem serves as an upper bound on where the optimal solution to the full problem might lie. The CLP approach found solutions for more students, although the Programmed method used less CPU time to come up with solutions. Both approaches appear to be close to giving the maximum capacity as the interval between them and the ILP relaxed solution is small.

5 The Placements Problem: version 2

After demonstrating the results of our initial solution to the clients. and discussing the allocations based on the initial model, the requirements inevitably changed. We received updated information on theatre capacities from the hospitals and it was necessary to refine the model in a number of ways.

5.1 Fluctuating capacity

In our initial model, we assumed that capacity was constant over time and identical for each phase. The updated data showed many instances where the capacity was reduced during certain timeslots and sometimes dependent on phase, so it was necessary to change the model so that the capacity as dependent also on phase and time. Hence we use $cap(h, sp, ph, t)$ instead of $cap(h, sp)$ in capacity constraint.

$$\forall h.\forall sp.\forall ph.\forall t.\, |st : alloc(st, t) = \langle h, sp, ph\rangle| \leq cap(h, sp, ph, t)$$

5.2 Flexible phase

Whereas we initially assumed that, for a given hospital and speciality, the number of placements available for surgery and anaesthetic are the same, in some cases the hospitals have given a limit on the total numbers in surgery + anaesthetic. Therefore this is a further choice that has to be resolved by the solver. We model this by considering that $cap(h, sp, ph, t)$ may itself be a variable to be solved rather than fixed in advance. For example, if hospital h, speciality sp and time t, the $cmax$ is available to either phase, we can impose a constraint.

$$cap(h, sp, phase1, t) + cap(h, sp, phase2, t) \leq cmax$$

In some cases, the hospitals have stipulated that the capacity cannot be split between phases, i.e. at any given time the capacity either all goes to anaesthetic or it all goes to surgery. To enforce this constraint, we can ensure that intermediate values are removed from the domains of the capacity variables:

$$\begin{array}{l} cap(h, sp, phase1, t) \in \{0, cmax\} \\ cap(h, sp, phase2, t) \in \{0, cmax\} \end{array}$$

5.3 Forced moves

After seeing the schedules we initially generated, the course organisers observed that some students did not visit more than one hospital, and this was considered undesirable. We then introduced a constraint requiring that students should move at least once. A constraint was added to force a move between the first and second placement.

$$\forall st.\exists h1.\exists h2.\left[\begin{array}{c} \exists sp.\exists ph.alloc(st, 1) = \langle h1, sp, ph\rangle \\ \wedge \\ \exists sp.\exists ph.alloc(st, 2) = \langle h2, sp, ph\rangle \\ \wedge \\ h1 \neq h2 \end{array}\right]$$

5.4 Extra specialities and combined specialities

After further consideration of the specialities available at the hospitals, course organisers chose to include all specialities which were deemed to be suitable as first year placements by the hospitals. The final list consisted of 12 possible specialities, some of which were combinations of the others, e.g. gynae and urology. The presence of the combined specialities makes the non-repetition constraint more complicated, as it is necessary to map each combined speciality to a set of atomic specialities, and test non-repetition on the atomic specialities.

5.5 Student preferences

In our initial model, we considered that each student had a fixed set of reachable hospitals, based on their home address. However, the ODP course organisers had instead asked each student to fill in a form selecting 3 hospitals in order of preference. The preferences were accounted for in the constraint solver. A value-ordering heuristic was introduced which orders possible placements according to preferred hospital order.

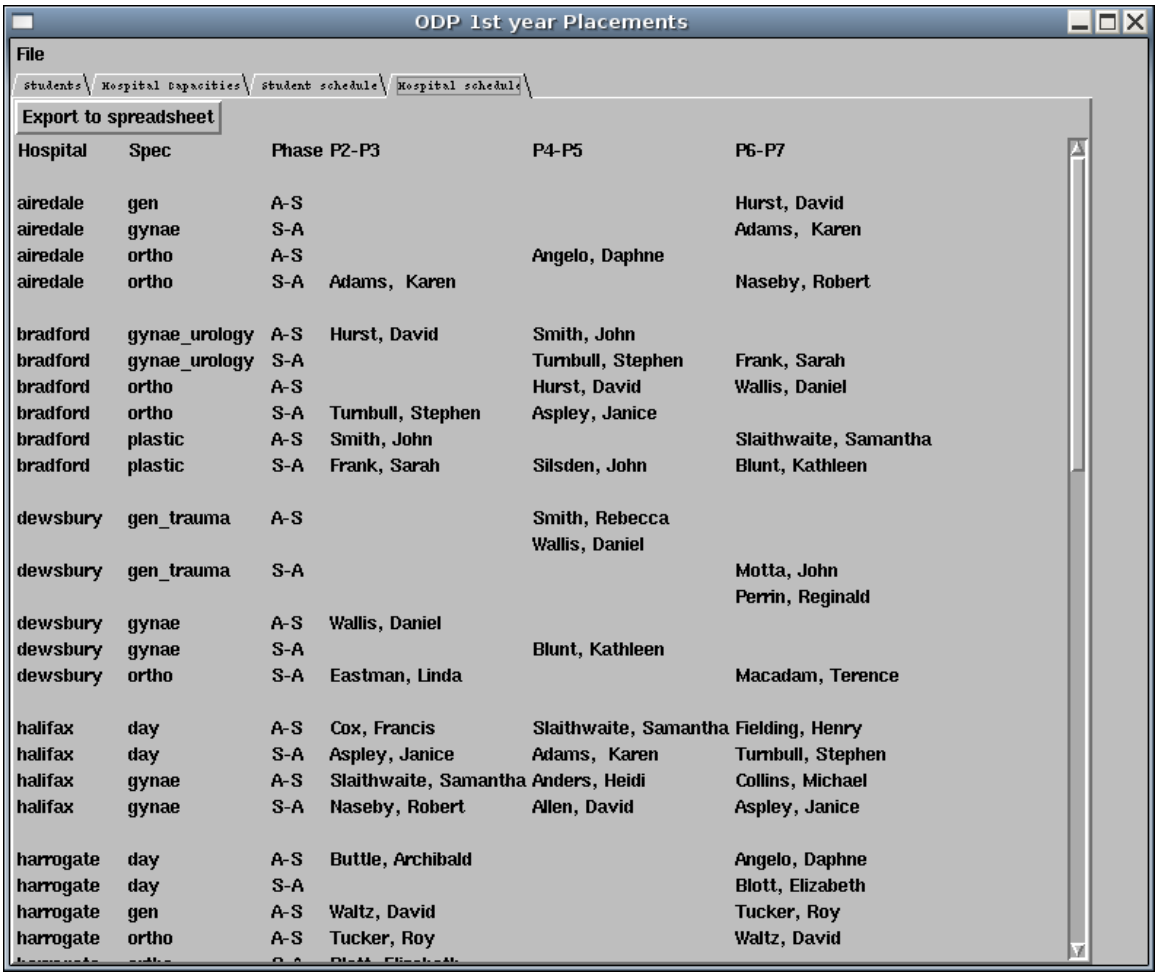
5.6 Even distribution

Our initial implementation included a variable-ordering heuristic but selected values naively according to a fixed and arbitrary order. This led to a bias towards certain hospitals and specialities, and hence an uneven allocation. As mentioned above, our enhanced system accounted for preferred hospitals in the value-ordering heuristic. In order to get a more even distribution between specialities, we used pseudo-random number generator to order the choice of specialities within the possible allocations at the preferred hospital. This gives an arbitrary sequence which is nevertheless deterministic and leads to a more even distribution.

5.7 Experimental Results

In our solution approaches to the first model of the problem, the Programmed and CLP solutions used similar heuristics. The Programmed approach makes an a priori ordering of students according to number of reachable hospitals, and the CLP program used a fail first heuristic, which dynamically selects the variable with smallest domain - i.e. the smallest choice of $\langle hosp, sp, ph \rangle$ tuples. In the CLP approach to the second model of the problem, we additionally included value ordering to bias in favour of student preferences, and a stochastic element to remove any bias in choosing specialities.

The CLP method was developed using the Oz platform to incorporate the constraints, with some success. The behaviour of the search procedure is that the heuristic often leads directly to a solution with little or no search. In cases where a significant amount of search took place, the solution was not found at all. It seems that the depth-first exploration of the search space is unsuccessful because the choices which are most important are those in the most constrained part of the problem, and these occur near the root of the search tree, hence we need a search procedure which is biased towards exploring alternatives near the root of the tree. We modified our search procedure by imposing a depth bound – complete search of the choices before the depth bound is allowed, but beyond the depth bound, the heuristic is obeyed entirely and no further choice points are generated. With this modification, solutions are found easily for a range of tests we have used. In the future, we plan to explore the



Hospital	Spec	Phase	P2-P3	P4-P5	P6-P7
airedale	gen	A-S			Hurst, David
airedale	gynae	S-A			Adams, Karen
airedale	ortho	A-S		Angelo, Daphne	
airedale	ortho	S-A	Adams, Karen		Naseby, Robert
bradford	gynae_urology	A-S	Hurst, David	Smith, John	
bradford	gynae_urology	S-A		Turnbull, Stephen	Frank, Sarah
bradford	ortho	A-S		Hurst, David	Wallis, Daniel
bradford	ortho	S-A	Turnbull, Stephen	Aspley, Janice	
bradford	plastic	A-S	Smith, John		Slaithwaite, Samantha
bradford	plastic	S-A	Frank, Sarah	Silsden, John	Blunt, Kathleen
dewsbury	gen_trauma	A-S		Smith, Rebecca Wallis, Daniel	
dewsbury	gen_trauma	S-A			Motta, John Perrin, Reginald
dewsbury	gynae	A-S	Wallis, Daniel		
dewsbury	gynae	S-A		Blunt, Kathleen	
dewsbury	ortho	S-A	Eastman, Linda		Macadam, Terence
halifax	day	A-S	Cox, Francis	Slaithwaite, Samantha	Fielding, Henry
halifax	day	S-A	Aspley, Janice	Adams, Karen	Turnbull, Stephen
halifax	gynae	A-S	Slaithwaite, Samantha	Anders, Heidi	Collins, Michael
halifax	gynae	S-A	Naseby, Robert	Allen, David	Aspley, Janice
harrogate	day	A-S	Buttle, Archibald		Angelo, Daphne
harrogate	day	S-A			Blott, Elizabeth
harrogate	gen	A-S	Waltz, David		Tucker, Roy
harrogate	ortho	A-S	Tucker, Roy		Waltz, David

Figure 3: Part of a Generated Hospital Schedule

use of more principled search algorithms which have similar properties, e.g. depth-bounded discrepancy search (DDS) [6].

The CLP method has been embedded in a user-friendly interface making data inport and export possible. The interface generates the schedules for hospitals and timetables for students. An example section of a completed hospital schedule is shown in fig. 3.

6 Related Work

In the Hospitals / Residents problem [4], residents are matched to hospitals subject to each hospital's preference order for residents, each resident's preference order for hospitals. and limited capacity of each hospital. The matching process is required to produce solutions which are stable in the sense that there exists no alternative pairing of a resident and a hospital which appears preferable to both parties [7].

A linear time algorithm exists for the simplest forms of the matching problem, allowing

[4]Thanks to our anonymous referees for pointing this out

it to be deployed on very large problem instances. In the US, a centralised system[5] is used to allocate internships for tens of thousands of medical graduates each year.

Although the application domain is related, our problem differs significantly from the Hospitals / Residents problem in that the preferences form only one among many constraints in our problem, and the preferences themselves only exist for one side – the hospitals do not rank the students. However, it may be useful to consider an analogous property of stability of our solutions.

In [8] the matching is defined as a constraint which would allow it to be integrated into a model alongside other constraints.

7 Conclusions and Future Work

We have tackled an allocation problem arising in a real application. A relatively simple constraint model of the problem has been successful, and has the potential to be useful to the ODP department, as well as other departments in universities offering similar medical courses, where the course requires a set of placements at nearby hospitals.

There is a lot more investigation that could be done to compare other solutions to the search problem. In particular we have not used student preferences to construct a cost function that would turn the problem into one of optimisation instead of finding any feasible solution. This is not a priority from the point of view of the users, because the approach we have produces satisfactory solutions quickly. It appears that the problem we have from the current hospital data is not in the hard region of the problem and this may be why a simple heuristic is successful. However, if the users want to use a significantly higher proportion of the available resources, the problem becomes more challenging and we may have to experiment with other techniques.

It would also be useful to return some analysis of the data – for instance the geographical distribution of demand and capacity. The system should check some small subproblems, e.g. with subsets of the students and subsets of the hospitals to detect whether any of these is unsolvable. This would help to discover infeasibility early and isolate an infeasible subproblem. One time-consuming aspect of the work has been in accommodating evolving requirements and building a system with a user-friendly graphical interface. One issue that we have not adequately addressed is how to improve the user interface to allow a mixed manual and automatic allocaton. For instance, the users would like to be able to freeze parts of the allocation and re-run automatic allocation on the remaining part. This is an mainly issue of user-interface design.

References

[1] Pascal van Hentenryck. *Constraint Satisfaction in Logic Programming.* M.I.T. Press, 1989.

[2] Peter Van Roy and Seif Haridi. *Concepts, Techniques and Models of Computer Programming.* M.I.T. Press, 2004.

[3] Robert M. Haralick and Gordon L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980.

[5] National Residents Matching Program, www.nrmp.org

[4] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[5] H. P. Williams. *Model Bulding in Mathematical Programming.* Wiley, 4th edition, 1999.

[6] Toby Walsh. Depth-bounded discrepancy search. In *IJCAI '97*, pages 1388–1395, 1997.

[7] D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 62:9–15, 1962.

[8] David F. Manlove, Greg O'Malley, Patrick Prosser, and Chris Unsworth. A constraint programming approach to the Hospitals / Residents problem. Technical Report TR-2005-196, Department of Computer Science, University of Glasgow, 2005.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman, 1979.