The Progressive Party Problem: Integer Linear Programming and Constraint Programming Compared

Barbara M. Smith
Division of Artificial Intelligence, School of Computer Studies,
University of Leeds, Leeds LS2 9JT

Sally C. Brailsford, Peter M. Hubbard and H. Paul Williams
Faculty of Mathematical Studies,
University of Southampton, Southampton SO9 5NH

Abstract

Many discrete optimization problems can be formulated as either integer linear programming problems or constraint satisfaction problems. Although ILP methods appear to be more powerful, sometimes constraint programming can solve these problems more quickly. This paper describes a problem in which the difference in performance between the two approaches was particularly marked, since a solution could not be found using ILP.

The problem arose in the context of organising a "progressive party" at a yachting rally. Some yachts were to be designated hosts; the crews of the remaining yachts would then visit the hosts for six successive half-hour periods. A guest crew could not revisit the same host, and two guest crews could not meet more than once. Additional constraints were imposed by the capacities of the host yachts and the crew sizes of the guests.

Integer linear programming formulations which included all the constraints resulted in very large models, and despite trying several different strategies, all attempts to find a solution failed. Constraint programming was tried instead and solved the problem very quickly, with a little manual assistance. Reasons for the success of constraint programming in this problem are identified and discussed.

1 Introduction

Discrete optimization problems of the kind that arise in many areas of operational research can be formulated as constraint satisfaction problems (CSPs). A CSP consists of a set of variables, each with a finite set of possible values (its domain), and a set of constraints which the values assigned to the variables must satisfy. In a CSP which is also an optimization problem, there is an additional variable representing the objective; each time a solution to the CSP is found, a new constraint is added to ensure that any future solution must have an improved value of the objective, and this continues until the problem becomes infeasible, when the last solution found is known to be optimal.

Many discrete optimization problems can be modeled using linear constraints and integer variables and thus formulated as integer linear programming problems. Operational Research has developed a battery of powerful techniques for solving such problems, but although the search algorithms available for solving CSPs are at first sight less powerful than ILP methods, sometimes constraint programming is a more successful approach (see [2, 6, 7]). It would be very useful to know which of these competing techniques to choose for a given problem, but the boundary between their areas of expertise has not yet been fully mapped. This paper describes a further example of a problem where constraint programming did much better than ILP; in fact, it proved impossible to solve the problem at all using ILP. The success of constraint programming in this case appears to be due to a number of factors in combination; these are discussed in section 8.

The problem is a seemingly frivolous one arising in the context of organising the social programme for a yachting rally. The 39 yachts at the rally were all moored in a marina on the Isle of Wight[1]; their crew sizes ranged from 1 to 7. To allow people to meet as many of the other attendees as possible, an evening party was planned at which some of these boats were to be designated hosts. The crews of the remaining boats would visit the host boats in turn for six successive half-hour periods during the evening. The crew of a host boat would remain on board to act as hosts; the crew of a guest boat would stay together as a unit for the whole evening. A guest crew could not revisit a host boat, and guest crews could not meet more than once. Additional capacity constraints were imposed by the sizes of the boats. The problem facing the rally organiser was that of minimising the number of host boats, since each host had to be supplied with food and other party prerequisites.

There were a number of complicating factors in the real-life problem. For example, the rally organiser's boat was constrained to be a host boat,

[1] off the south coast of England.

although it had a relatively small capacity, because he had to be readily available to deal with emergencies. Two other boats had crews consisting of parents with teenage children, and these boats were also constrained to be host boats; the crews split up so that the parents remained on board the host boat and the children became a "virtual boat" with capacity of zero. The rally organiser's children formed a third virtual boat, giving 42 boats altogether. The data for this problem is given in Table 1.

Boat	Capacity	Crew	Boat	Capacity	Crew	Boat	Capacity	Crew
1	6	2	15	8	3	29	6	2
2	8	2	16	12	6	30	6	4
3	12	2	17	8	2	31	6	2
4	12	2	18	8	2	32	6	2
5	12	4	19	8	4	33	6	2
6	12	4	20	8	2	34	6	2
7	12	4	21	8	4	35	6	2
8	10	1	22	8	5	36	6	2
9	10	2	23	7	4	37	6	4
10	10	2	24	7	4	38	6	5
11	10	2	25	7	2	39	9	7
12	10	3	26	7	2	40	0	2
13	8	4	27	7	4	41	0	3
14	8	2	28	7	5	42	0	4

Table 1: The data

2 The Uncapacitated Problem

If we ignore the capacity constraints, just one host boat can accommodate any number of guest boats for one time period. For more than one time period, we can easily find a lower bound on the number of hosts required from the following argument. If g guest crews visit host i at time 1, then there must be at least g other hosts to accommodate them in the following time period. (The guests cannot visit host i again, and must visit g different hosts so as not to meet another crew again.) In fact, the required $g+1$ hosts could each accommodate up to g visiting guest crews at time 1, without the guest crews meeting again at time 2, giving $g(g+1)$ guest crews in total. For more than 2 time periods, $g(g+1)$ is clearly an upper bound on the number of guest crews that $g+1$ hosts can accommodate. For instance, 6 hosts can accommodate at most 30 guest boats; 7 hosts can accommodate at most 42. In fact, these limits can be attained (still assuming no constraints on the hosts' capacities, and provided that the number of time periods is not greater than the number of hosts, in which case it becomes impossible for guest crews not to visit the same host more than once), so that with 42 boats in all, we need 7 to be hosts (and therefore 35 to be guests).

However, for the real-life problem, the capacity constraints are binding and the number of host boats required is at least 13, as shown in Section 3.

3 A Lower Bound

A lower bound on the number of hosts required, taking into account the capacity constraints, was found by using linear programming to solve a considerable relaxation of the original problem. This simply required that the guest crews, as a whole, should fit into the total spare capacity of the host boats[2] for one time period.

The same lower bound can alternatively be found from a simple argument: a necessary condition for feasibility is that the total capacity of the host boats is not less than the total crew size of all the boats. The smallest number of hosts that meet this condition is therefore found by ordering the boats in descending order of total capacity. With this ordering, the first 13 boats can accommodate all the crews; the first 12 boats cannot.

This suggests that in general the host boats should be chosen in descending order of total capacity. However, this heuristic was not arrived at until after the linear programming work on the problem had been completed, partly because it seemed counter-intuitive that the crew sizes should be ignored when selecting the hosts. Moreover, maximising the number of spare places is not the only consideration when selecting the host boats, since each crew has to stay together. Provided that the total capacity of the hosts is large enough, the choice of hosts may need to consider the spare capacity of each boat and how well different crew sizes fit into it.

Hence the model described below includes the selection of the host boats, even though in practice the choice of hosts was in large part guided by heuristics.

4 Integer Programming Approach

4.1 First Formulation

The first attempt at finding an optimal solution was made at the University of Southampton, where the problem was formulated as a zero-one integer programme. The variables are: $\delta_i = 1$ iff boat i is used as a host boat, and $\gamma_{ikt} = 1$ iff boat k is a guest of boat i in period t. (The rally organiser's boat was constrained to be a host in all models.)

As mentioned in Section 1, the objective was to minimise the number of hosts:

[2] i.e. the remaining capacity after accommodating the host crews themselves.

$$\text{minimise } \sum_i \delta_i \qquad \text{subject to:}$$

Constraints CD. A boat can only be visited if it is a host boat.

$$\gamma_{ikt} - \delta_i \leq 0 \qquad \text{for all } i,k,t; i \neq k$$

Constraints CCAP. The capacity of a host boat cannot be exceeded.

$$\sum_{k,k\neq i} c_k \gamma_{ikt} \leq K_i - c_i \qquad \text{for all } i,t$$

where c_i is the crew size of boat i and K_i is its total capacity.
Constraints GA. Each guest crew must always have a host.

$$\sum_{i,i\neq k} \gamma_{ikt} + \delta_k = 1 \qquad \text{for all } k,t$$

Constraints GB. A guest crew cannot visit a host boat more than once.

$$\sum_t \gamma_{ikt} \leq 1 \qquad \text{for all } i,k; i \neq k$$

Constraints W. Any pair of guests can meet at most once.

$$\gamma_{ikt} + \gamma_{ilt} + \gamma_{jks} + \gamma_{jls} \leq 3 \qquad \begin{array}{c}\text{for all } i,j,k,l,t,s; \\ i \neq j; i \neq k; k < l; i \neq l; \\ k \neq j; j \neq l; s \neq t\end{array}$$

The constraints W, which have six indices, clearly lead to a huge number of rows when the problem is modelled. The number of rows is $O(B^4T^2)$, where B is the number of boats and T is the number of time periods. However, this model has a moderate number of variables, namely $O(B^2T)$.

The size of the problem was reduced by taking account of the fact that in any optimal solution there are some boats which will always be chosen to be hosts because of their large capacity and small crew size. By the same token, some boats would clearly never be chosen to be hosts: for example, the three virtual boats with zero capacity. The data was ordered by decreasing (total capacity − crew size) and parameters *hostmin* and *hostmax* were introduced, such that the range of indices for potential hosts was restricted to 1, .. , *hostmax* and the range of indices for potential guests was restricted to *hostmin*+1, .. , 42.

The formulation was tested on a reduced problem with 15 boats and 4 time periods, and with $hostmin = 4$ and $hostmax = 8$. This resulted in a model with 379 variables and 18,212 rows. The LP relaxation solved in 259 seconds using the XPRESSMP optimiser[3] on an IBM 486 DX PC, in 816 simplex iterations.

[3]XPRESS MP (Version 7). Dash Associates, Blisworth House, Blisworth, Northants NN7 3BX, U.K.

4.2 Second Formulation

To reduce the number of constraints in the previous formulation, a further set of zero-one variables was introduced:

$$x_{iklt} = 1 \text{ iff crews } k \text{ and } l \text{ meet on boat } i \text{ in time period } t$$

and the constraints W were replaced by the three following sets S, V and Y. S and V together define the x variables in terms of the γ variables:
Constraints S.

$$2x_{iklt} - \gamma_{ikt} - \gamma_{ilt} \leq 0 \qquad \text{for all } i,k,l,t; k < l; i \neq k; i \neq l$$

Constraints V.

$$\gamma_{ikt} + \gamma_{ilt} - x_{iklt} \leq 1 \qquad \text{for all } i,k,l,t; k < l; i \neq k; i \neq l$$

and constraints Y then replace constraints W in the first formulation:
Constraints Y. Any pair of guest crews can meet at most once.

$$\sum_t \sum_{l,l>k} x_{iklt} \leq 1 \qquad \text{for all } i,k$$

The number of variables is now increased to $O(B^3T)$, but the number of rows is reduced, also to $O(B^3T)$.

5 Experiments on A Reduced Problem

The second formulation was used in a variety of computational experiments with the reduced 15-boat problem. As before, $hostmin = 4$ and $hostmax = 8$. Firstly, the problem was solved directly (model PS1). This gave an optimal solution with 5 hosts in a total time of 2214 secs. This was used as a basis for comparison in several experiments.

First, a facility of the XPRESSMP package was used which enables certain constraints to be introduced only if a particular solution violates them. This greatly reduces the initial size of a model. This facility is called MVUB (Make Variable Upper Bounds) and applies only to constraints of the form $x - My \leq 0$. The CD constraints were in this form already and the S constraints could be disaggregated to get them into the proper form, giving:

$$x_{iklt} - \gamma_{ikt} \leq 0 \text{ and } x_{iklt} - \gamma_{ilt} \leq 0$$

Normally disaggregation would result in a tighter LP relaxation, but since in this case all the coefficients of x_{iklt} in the other constraints of the model are unity, it can be shown by Fourier-Motzkin elimination that this will not

be the case [8]. In the second version of the model (PS11) both the CD and the S constraints were modelled using MVUB; in the third (PS12) the S constraints were modelled explicitly and the CD constraints were modelled using MVUB. The results are shown in Table 2; the total solution time for PS1 was less than for either of the new versions. Thus the MVUB feature was not helpful in this case.

Model	PS1	PS11	PS12
Rows	4386	2130	4022
Columns	2271	2271	2271
LP solution time (secs)	101	16	19
Number of iterations	1474	696	497
LP objective value	3.42	3.42	3.42
MVUB time (secs)	*n.a.*	561	697
Branch-&-Bound time (secs)	2113	1852	3789
Number of nodes	287	279	311
IP objective value	5.00	5.00	5.00

Table 2: Results with MVUB

Next, special ordered sets of type I were tried. A set of variables form a special ordered set of type I if at most one of them can be nonzero. For example, the γ variables could be treated as special ordered sets: for each value of i and k, at most one of the set $\{\gamma_{ikt}, t = 1, .., 6\}$ can be nonzero. This device is useful if there is some natural ordering on the variables, when it can reduce the time spent doing branch-and-bound. However, in this case there is no natural ordering, since the time periods are interchangeable: in any solution, periods 1 and 6, say, can be swapped and the solution will still be valid. This meant that the approach was not helpful and in fact made branch-and-bound slower.

It would also be possible to tighten the LP relaxation by adding extra "covering" constraints generated from the CCAP constraints. For example, from the capacity constraint

$$\gamma_{121} + 2\gamma_{131} + 4\gamma_{141} + 3\gamma_{151} \leq 7$$

the following covering constraints could be derived:

$$\gamma_{121} + \gamma_{141} \leq 1$$

$$\gamma_{121} + \gamma_{131} + \gamma_{151} \leq 2$$

$$\gamma_{121} + \gamma_{141} + \gamma_{151} \leq 2$$

However, there would be a vast number of such constraints and so this did not seem a particularly fruitful approach.

Another approach was to omit the S, V and Y constraints, solve the LP relaxation of the resulting problem and then add in cuts of the form

$$\sum_{i \in Q} \gamma_{kit} - \sum_{i,j \in Q} x_{kijt} \leq \left\lfloor \frac{\mid Q \mid}{2} \right\rfloor$$

for index sets Q, where $\mid Q \mid$ is odd. By inspection, many of these were violated by the fractional solution. However, automating the process of inspecting the solution, identifying the appropriate index sets and then generating the corresponding cuts would have been computationally prohibitive. Equally, there would be no advantage in generating all possible cuts (even for sets of cardinality 3 only), as this would simply have resulted in another enormous model.

To summarise, the experiments with the reduced problem did not indicate a successful solution strategy for the full problem.

6 Experiments on The Full Problem

The size of the full model defeated all the available modellers, even using indices restricted by *hostmin* and *hostmax*. Therefore, a heuristic approach was adopted, based on the recognition that the total capacity of the first 13 boats (arranged, as described earlier, in decreasing order of spare capacity) was sufficient to accommodate all the crews (there would be 4 spare places), and that the largest guest crew could be accommodated on all but three of the hosts. Hence if a solution with 13 hosts was possible, these 13 hosts seemed a reasonable choice[4].

A solution with 14 hosts was found, by relaxing the meeting constraints and specifying that at least the first 14 boats, and at most the first 15, had to be hosts. An integer solution was found in 598 secs. There were only a few violations of the meeting constraints and, by manually adding in the violated constraints, a feasible solution to the original problem was found.

It began to seem that this might be an optimal solution. Therefore the first 13 boats were fixed as hosts and an attempt was made to prove that this was infeasible. The indices of the guest boats in the meeting constraints were restricted to 21 to 42 (simply because this gave the largest model that XPRESSMP could handle). The model was still large by most standards: 19,470 constraints and 11,664 variables. It was run using a parallel implementation of OSL[5] on seven RS/6000 computers at IBM UK Scientific Centre,

[4]As described in Section 3, it was later realised that the first 13 boats in order of total capacity, K_i, would give a larger number of spare places after all the guests had been accommodated. Nevertheless, the problem was in theory feasible, in terms of total capacity, with the 13 selected hosts.

[5]Optimization Subroutine Library (OSL), IBM Corporation.

Hursley. The run was aborted after 189 hours, having failed to prove infeasibility. OSL had processed about 2,500 nodes, of which around 50% were still active: 239 were infeasible. Some nodes were taking over two hours to evaluate.

7 Constraint Programming Approach

This alternative approach was suggested by the desire to prove infeasibility for the 13-host model, in the light of the failure of OSL. However, it turned out that constraint programming was able to find a feasible solution very rapidly, albeit with a little manual intervention. The work was carried out at the University of Leeds.

The progressive party problem, with the 13 specified host boats, was formulated as a constraint satisfaction problem (CSP) and implemented in ILOG Solver [5], a constraint programming tool in the form of a C++ library. Solver has a large number of pre-defined constraint classes and provides a default backtracking search method, for solving CSPs. If necessary, new constraint classes and search algorithms can be defined, but these facilities were not required in this case. The default search algorithm is an implementation of full lookahead, as described by Haralick and Elliott [3]. The algorithm repeatedly chooses an unassigned variable, chooses a value for that variable from its current domain and makes a tentative assignment. The constraints are then used to identify the effects of the assignment on future (still unassigned) variables, and any value in the domain of a future variable which conflicts with the current assignment (and the rest of the current partial solution) is temporarily deleted. Furthermore, the subproblem consisting of the future variables and their remaining domains is made arc consistent, which may result in further values being deleted. If at any stage the domain of a future variable becomes empty, the algorithm backtracks and retracts the last assignment. The algorithm can therefore be viewed as the forward checking algorithm, a commonly-used algorithm for solving CSPs (also described in [3]), with additional constraint propagation to re-establish arc consistency after each assignment.

7.1 CSP Formulation

The first advantage of the constraint programming approach was that the formulation as a CSP was much more compact than had been previously possible. Since the task in this case was to show (if possible) that the problem with 13 specific host boats was infeasible, host and guest boats were treated separately in the formulation. Suppose that there are G guest boats, H host boats and T time periods.

The principal variables, h_{it}, represent the host boat that guest boat i visits at time t; the domain of each h_{it} is the set {1,.., H}, and there are GT such variables. The constraints that every guest boat must always have a host and that in any time period a guest boat can only be assigned to one host are automatically satisfied by any solution to the CSP, which must have exactly one value assigned to each h_{it}, i.e. exactly one host assigned to each guest boat in each time period.

The constraints that no guest boat can visit a host boat more than once are expressed in the CSP by:

$$h_{i1}, h_{i2}, h_{i3}, .., h_{iT} \text{ are all different} \qquad \text{for all } i$$

For each i, this gives a single Solver constraint, equivalent to $T(T-1)/2$ binary not-equals constraints, i.e. $h_{i1} \neq h_{i2}$, etc.

The capacity constraints are dealt with, as in the LP, by introducing new constrained 0-1 variables, corresponding to the γ variables of the LP formulation: $v_{ijt} = 1$ iff guest boat i visits host j at time t. The relationships between these variables and the h_{it} variables are specified by GHT Boolean constraints:

$$v_{ijt} = 1 \text{ iff } h_{it} = j \qquad \text{for all } i, j, t$$

and as in the LP, the capacity constraints are then:

$$\sum_i c_i v_{ijt} \leq C_j \qquad \text{for all } j, t$$

where c_i is the crew size of guest boat i and C_j is the spare capacity of host boat j, after accommodating its own crew.

The constraints that no pair of crews can meet twice also require the introduction of a new set of 0-1 variables: $m_{klt} = 1$ iff crews k and l meet at time t. The constraints linking the new variables to the original h variables are:

$$\text{if } h_{kt} = h_{lt} \text{ then } m_{klt} = 1 \qquad \text{for all } k, l, t; k < l$$

The meeting constraints are expressed by:

$$\sum_t m_{klt} \leq 1 \text{ for all } k, l; k < l$$

Because the m variables have only three subscripts, rather than four as in the equivalent (x) variables in the LP, the CSP has only $O(B^2T)$ variables and $O(B^2T)$ constraints.

7.2 Symmetry Constraints

The constraints just described are sufficient to define the problem: a number of additional constraints were introduced to reduce the symmetries in the problem, as much as possible. For any solution, there are many equivalent solutions, which have, for instance, two guest boats with the same size crew interchanged throughout. Such symmetries in the problem can vastly increase the size of the search space and so the amount of work that has to be done. If there are no solutions, searching through many equivalent partial solutions can be extremely time-consuming. Symmetry can be avoided, or at least reduced, by adding constraints to eliminate equivalent solutions. (Puget [4] discusses this approach to avoiding symmetry.)

First, an ordering was imposed on the time-periods, which are otherwise interchangeable: the first guest boat must visit the host boats in order. (As described in the next section, the first guest boat was the one with the largest crew.)

The second set of constraints distinguishes between guest boats with the same size crew: if i, j are such a pair, with $i < j$, then for the first time period, we impose the constraint:

$$h_{i1} \leq h_{j1}$$

To allow for the fact that both boats may visit the same host at time 1 (i.e. $h_{i1} = h_{j1}$), but if so, they must visit different boats at time 2:

$$\text{either } h_{i1} < h_{j1} \text{ or } h_{i2} < h_{j2}$$

Finally, constraints were added to distinguish (to an extent) between host boats with the same spare capacity: if j and k are two such host boats, with $j < k$, the first guest boat cannot visit host k unless it also visits host j.

7.3 Solving the Problem

It is next necessary to choose variable and value ordering heuristics, i.e. rules for deciding which variable to consider next and which value to assign to it. Although the formulation just described has a great many variables, assigning values to the h_{it} variables is sufficient to arrive at a solution, and in devising variable and value ordering heuristics only these principal variables need be considered. Good variable ordering heuristics, in particular, are often crucial to the success of constraint programming methods. A heuristic which is commonly used is based on the "fail-first" principle, that is, choose the variable which is likely to be hardest to assign. In the forward checking algorithm, this means choosing the variable with the smallest remaining domain; ties may be broken by choosing the variable involved in most constraints.

Finally, variables are considered in the order in which they are defined; to give priority to the largest crews, the guest crews, and so the corresponding h_{it} variables, were arranged in descending order of size.

In ordering the values, a general principle is to choose first those values which seem most likely to succeed, and the host boats accordingly were considered in descending order of spare capacity.

The problem formulation was first tested on smaller problems than the full 13 hosts, 29 guests, 6 time periods problem. Several smaller problems were solved very quickly (in 1 or 2 seconds on a SPARCstation IPX), with little backtracking, and the program was shown to be producing correct solutions. However, the full size problem ran for hours without producing any result.

It was then decided to assign all the variables relating to one time period before going on to the next. Hence, each time period was solved separately, but the solutions for each time period constrained future solutions. In effect, this was another variable ordering heuristic, taking priority over the others; however, the program was not able to backtrack to earlier time periods. The plan was to find a solution for as many time periods as could be solved within a short time, and then to print out the domains of the remaining variables. It was hoped that this would give some clue as to why the program could not proceed. With this modification, the program found a solution for five time periods very quickly (which it had not previously been able to do). At this point, the domain of any variable corresponding to the 6th time period contains those hosts that the corresponding boat can visit and has not already visited. An attempt was made to fit the guest crews into those host boats which they could still visit, by hand, in order to see why it could not be done. However, a solution was found which appeared to be feasible; adding some of the assignments to the program as extra constraints confirmed that a solution based on these assignments did obey all the constraints.

Hence, the 13 hosts, 29 guests, 6 time periods problem, which had been thought to be insoluble, had been solved, though with some manual assistance. An optimal solution to the original problem had therefore been found.

Subsequently, the extra constraints on the 6th time period were removed, and the program allowed to search for a solution without this intervention; it found a solution in 27 minutes, and went on to find a solution for the 7th time period in another minute, so that the party could have lasted for longer without requiring more hosts!

8 Discussion

For this particular problem, constraint programming, using constraint propagation and full-lookahead, succeeded spectacularly in finding a solution very quickly where linear programming had failed to find a solution at all. More-

over, on those problems which linear programming succeeded in solving, constraint programming found solutions much more quickly. A number of reasons to account for its success in this case can be identified.

8.1 Compactness of representation

The fundamental difficulty with linear programming appears to lie in finding a compact representation of the problem. The formulations described earlier show that the CSP representation requires far fewer constraints and variables than the ILP. This is possible because of the greater expressive power of the constraints allowed in constraint satisfaction problems, which are not restricted to linear inequalities as in linear programming. In turn, the greater expressiveness is allowed by the fact that constraint satisfaction algorithms make relatively unsophisticated (although very effective) use of the constraints, compared with the simplex algorithm, for instance; it is only necessary to be able to detect whether a particular set of assignments satisfies a given constraint or not.

Furthermore, although the total number of variables and constraints is important in constraint programming, it commonly happens, as here, that in modelling a complex combinatorial problem as a CSP, there is a set of principal variables together with subsidiary variables which are introduced for the purpose of modelling the constraints. Typically, the forward checking algorithm is applied only to the principal variables of the problem. As already mentioned, in this case the solution is found by assigning values to the h_{it} variables in turn, i.e. by assigning a host to each guest boat i for each time t. The subsidiary variables are automatically instantiated in this process, because of the constraints linking them to the principal variables, and they are used in effect as vehicles for propagating the problem constraints to the domains of other principal variables. Hence, the effective complexity of the problem is less than the total number of constraints and variables suggests. It is still, however, a large problem, having 29×6 variables, each with 13 possible values, giving $13^{29\times6}$ possible assignments.

8.2 Constraint propagation

The constraints in the progressive party problem are such that, using a CSP formulation and a lookahead algorithm, the effect of any assignment of a value to a variable can usually be propagated immediately to the domains of related variables. For instance, as soon as an assignment is made to an h_{it} variable, i.e. a host is assigned to guest crew i at time t, the same value can be removed from the domain of any variable corresponding to a crew that has already met crew i. The capacity constraints are the only ones that may not immediately prune the domains of other principal variables. However, given

the capacities and crew sizes, the maximum number of guest crews that can visit a host simultaneously is 5, and in practice the number is almost always 3 or less. An assignment to an h_{it} variable will, therefore, very often result in the capacity constraints being used to prune other domains. Since the capacity constraints are binding, it is important that infeasible assignments should be detected as early as possible in this way. In other problems, by contrast, constraints on resources may only have any pruning effect once most of the variables involved in the constraint have been instantiated; this can lead to a large amount of searching before a set of assignments satisfying the constraint is found.

8.3 Solution Strategy

The attempts detailed in section 5 to find a good solution strategy using ILP are based on the mathematical properties of the model and not on the original problem. When it succeeds, this is, of course, one of the strengths of linear programming: the methods that have been developed are independent of the specific problem being addressed. However, in this case, where the attempts failed, it seems a disadvantage that the important features of the original problem cannot be used to direct the search for a solution. Indeed, the ILP model makes it difficult to see that the problem is essentially one of assigning a host boat to each guest boat in each time period: the majority of the variables are x variables, introduced solely to model the meeting constraints, and the fact that each guest boat must be assigned to exactly one host boat at any time is expressed only implicitly in the constraints.

In the CSP formulation, on the other hand, it is easy to see the essentials of the problem, because the principal variables represent precisely the assignment of a host boat to each guest boat in each time period. This allows a solution strategy to be devised around reasoning about these assignments. Although this approach may seem very problem specific, it requires only that variable and value ordering strategies be defined, and often, as here, general principles apply; choose next the variable which is likely to be hardest to assign, and choose a value for it which is likely to succeed. Solving the problem separately for each time period is admittedly a more problem-specific heuristic, but it is an example of an approach worth trying when a problem can be naturally divided into subproblems, and very quick and easy to implement.

8.4 Proof of optimality

Finally, it is easy to show that a solution with 12 hosts is impossible, from the fact that the capacity constraints cannot be met even for a single time period; hence, when a solution with 13 hosts was found, it was known to be optimal. In other situations, proving optimality can be much more difficult.

8.5 The Role of Heuristics

All these factors, together with a degree of good luck in the choice of heuristics, combined to make what was a very difficult problem for linear programming, a tractable one for constraint programming. Even so, the size of the problem meant that it was still potentially too difficult for constraint programming to solve if a great deal of search was required.

The role of heuristics in finding a solution is clearly crucial; for instance, the program was unable to find a solution with 13 hosts when all the time periods were considered together. Considering each time period in turn, as it was implemented in this case, has obvious limitations; because backtracking to earlier time periods is not allowed, this strategy is not guaranteed to find a solution if there is one, and it cannot show that a problem is infeasible. In general, it may be necessary to experiment with different combinations of heuristics on a given problem instance in order to get a good solution. Even then, it may be difficult to find a solution: a modification of the original problem, to make the individual crew sizes much more equal while keeping the total size the same, has proved much more difficult (even for one time period, where a solution can easily be found manually).

Ironically, it seems very probable that if the 13-host problem had indeed been infeasible, as originally supposed, the constraint programming approach would not have been able to prove infeasibility: although it is easy to show that a solution with 12 hosts is impossible, because the capacity constraints cannot be met even for a single time period, a problem which is 'only just' infeasible, because the meeting and capacity constraints cannot be simultaneously satisfied for the required number of time periods, would require a complete search of a very large search space, and would be extremely difficult to prove infeasible.

9 Related Work

Papers which also compare integer linear programming and constraint programming applied to particular problems are [2, 7]. Van Hentenryck and Carillon [7] describe a warehouse location problem, and suggest that the ILP model, because it has a great many variables relating to the allocation of customers to warehouses, disguises the fact that the essence of the problem is to decide which of the possible warehouse locations should be chosen. The constraint programming approach, on the other hand, is based on reasoning about the warehouses. This is similar in some respects to the progressive party problem, which also has a large number of additional ILP variables to model the meeting constraints, as described in section 8.3. However, in the warehouse location problem, the ILP and CSP models have an identical set

of 0-1 variables representing whether each warehouse is to be used or not, so that the difficulty in the ILP is that the main variables are swamped by other variables. In the progressive party problem, an additional difficulty is that the ILP, unlike the CSP, has no variables representing directly the allocation of a host boat to each guest boat in each time period.

The paper by Dincbas, Simonis and van Hentenryck [2] discusses a case in which the expressive power of the constraints in constraint programming allows a radical reformulation of the obvious ILP model, giving a much smaller problem. A formulation with n variables, each with m values, and constraints which are linear inequalities, is expressed instead in terms of m variables each with n values, and more complex constraints. This changes the number of possible assignments of values to variables from m^n to n^m. Since the number of possible assignments indicates the total size of the search space, this is an advantage if m is much smaller than n. For instance, in [2], a CSP with complexity 4^{72} is reformulated to give a problem with complexity 72^4.

However, this is not the reason for the success of constraint programming in the progressive party problem: in that case, the CP formulation still has a relatively small number of values compared with the number of variables, and the number of possible assignments is $13^{29\times6}$. In theory, therefore, we should consider reversing the formulation in some way, making the host boats the variables. However, in this case reformulation is not a sensible option. One complication is the time dimension: the variables would have to correspond to each host boat in each time period, giving 13×6 variables in all. The values would then be the possible combinations of guest boats which could be assigned to each variable, and there are a great many such combinations; the constraints would also be very difficult to express. So although reversing the formulation can be extremely valuable in some cases, reducing a large problem to a smaller problem which can be solved much more quickly, it is not possible in this case.

A recent paper by Puget and De Backer [6] compares integer linear programming and constraint programming in general. They conclude that a crucial factor is the degree of propagation that the constraints of a problem allow: if each assignment of a value to a variable can be expected to trigger the pruning of many values from the domains of other variables, so that large parts of the search space do not have to be explored, constraint programming can be expected to be successful. As discussed in section 8.2, the constraints in the progressive party problem are very effective in propagating the effects of assignments to other variables. In other cases, for instance, where the constraints involve large numbers of variables, constraint propagation may be much less useful, and if the problem can be naturally represented by linear constraints, integer linear programming may be more efficient. Beringer and De Backer [1] discuss the combination of the two approaches for optimisation problems, where the bounds given by applying the simplex method

to a linear relaxation of the problem can provide useful information which would not otherwise be available. On suitable problems, for instance multi-knapsack problems, allowing the two approaches to co-operate gives much faster solution times than could otherwise be achieved.

10 Conclusions

Although the progressive party problem may not be a practical problem, except for members of yacht clubs, it has many of the classical features of combinatorial optimization problems, and was expected to be amenable to linear programming techniques. However, as we have shown, the resulting models were too large to be solved, whereas constraint programming found an optimal solution very quickly.

The success of constraint programming in solving this problem is due to a combination of factors, discussed in section 8. Some of these reasons have been identified in other studies of problems where constraint programming out-performed ILP, as discussed in section 9. However, unlike the previous studies, in this problem both models turned out to be extremely large; ILP failed because the model was too large to be solved, but also, it would not have been possible to explore the complete search space arising from the constraint programming formulation. In practice, many real problems are too large to be able to guarantee to find a solution; this paper shows that even so, constraint programming can succeed through careful choice of heuristics to direct its search.

Our experience with this problem suggests that constraint programming may do better than integer linear programming when the following factors are present:

- The problem cannot easily be expressed in terms of linear constraints: constraint programming will then give a more compact representation.
- The constraints allow the early propagation of the effects of assignments to the domains of other variables. This happens if each constraint involves only a small number of variables, but also sometimes with global constraints, as in this case: the capacity constraints are triggered after only a small number of guest boats have been assigned to the same host at the same time.
- It is easy to devise good solution strategies for the problem and hence take advantage of the fact that the constraint programming formulation represents the problem much more directly that the ILP formulation typically does.

- A tight bound on the value of the objective in an optimal solution is available, so that if an optimal solution is found, it can be recognised as such (unless, of course, the problem is sufficiently small to be able to prove optimality by doing a complete search).

Further comparisons between the two approaches are still needed to quantify some of these factors and to give a clearer idea of when constraint programming should be chosen in preference to integer linear programming.

Acknowledgement

We are very grateful to William Ricketts of IBM UK Scientific Centre, Hursley Park, Winchester, for his enthusiastic help with the computational experiments on the large LP model.

References

[1] H. Beringer and B. De Backer. Combinatorial Problem Solving in Constraint Logic Programming with Cooperating Solvers. In C. Beierle and L. Plumer, editors, *Logic Programming: Formal Methods and Practical Applications*, chapter 8, pages 245–272. Elsevier Science B.V., 1995.

[2] M. Dincbas, H. Simonis, and P. van Hentenryck. Solving a Cutting-Stock problem in constraint logic programming. In R. Kowalski and K. Brown, editors, *Logic Programming*, pages 42–58. 1988.

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

[4] J.-F. Puget. On the Satisfiability of Symmetrical Constrained Satisfaction Problems. In *Proceedings of ISMIS'93*, 1993.

[5] J.-F. Puget. A C++ Implementation of CLP. In *Proceedings of SPICIS94 (Singapore International Conference on Intelligent Systems)*, 1994.

[6] J.-F. Puget and B. De Backer. Comparing Constraint Programming and MILP. In *APMOD'95*, 1995.

[7] P. van Hentenryck and J.-P. Carillon. Generality versus Specificity: an Experience with AI and OR techniques. In *Proceedings of AAAI-88*, volume 2, pages 660–664, 1988.

[8] H. P. Williams. The elimination of integer variables. *JORS*, 43:387–393, 1992.