Constraints, An International Journal, 1, 119–138 (1996)
© 1996 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

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

BARBARA M. SMITH bms@scs.leeds.ac.uk
Division of Artificial Intelligence, School of Computer Studies, University of Leeds, Leeds LS2 9JT

SALLY C. BRAILSFORD scb@maths.soton.ac.uk
PETER M. HUBBARD pmh@maths.soton.ac.uk
H. PAUL WILLIAMS h.p.williams@soton.ac.uk
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 organizing 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.

Keywords: combinatorial optimization, integer linear programming, constraint programming

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* X = $\{x_1, ..., x_n\}$;
- for each variable x_i, a finite set D_i of possible values (its *domain*);
- and a set of *constraints* restricting the values that the variables can simultaneously take.

Formally, a possible constraint $C_{ijk...}$ between the variables $x_i, x_j, x_k, ...$ is a relation on the corresponding domains, i.e. $C_{ijk...} \subseteq D_i \times D_j \times D_k \times .....$: the constraint is a subset of the possible combinations of values of the relevant variables, representing the assignments that can be simultaneously made to these variables. A solution to a CSP is an assignment of a value from its domain to every variable, in such a way that every constraint is satisfied.

An optimization problem may be modelled as a CSP by adding a 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.

In many discrete optimization problems the constraints can be expressed as linear inequalities or equations, and thus the problems can be formulated as integer linear programming problems. Operational Research has developed a battery of powerful techniques for solving such problems, which make use of the special form of the constraints. However, 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 (Dincbas *et al.*, 1988, Puget & De Backer, 1995, van Hentenryck & Carillon, 1988)). 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 9.

The problem is a seemingly frivolous one arising in the context of organizing the social programme for a yachting rally. The 39 yachts at the rally were all moored in a marina on the Isle of Wight (off the south coast of England); 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 organizer was that of minimizing 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 organizer's boat was constrained to be a host boat, 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 organizer's children formed a third virtual boat, giving 42 boats altogether. The data for this problem is given in Table 1.

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

Table 1. The boat and crew data

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

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. Hence, with 42 boats in all, we should 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 (i.e. the remaining capacity after accommodating the host crews themselves) 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 (after the boats 1 to 3 in

Table 1, which must be host boats). 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 (and was not used for any of the experiments reported in this paper), partly because it seemed counter-intuitive that the crew sizes should be ignored when selecting the hosts. Moreover, maximizing the capacity is not the only consideration when selecting the host boats, since each crew has to stay together as a unit. 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 organizer's boat was constrained to be a host in all models.)

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

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

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.

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

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. After the first three boats in Table 1, the remaining boats were ordered by decreasing (total capacity − crew size) and parameters *hostmin* and *hostmax* were introduced, such that the range of indices in the new ordering 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 optimizer[1] on an IBM 486 DX PC, in 816 simplex iterations.

4.2. *Second Formulation*

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

$x_{iklt} = 1$ iff crews k and l meet on boat i 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.

$$\sum_t \sum_i x_{iklt} \leq 1 \qquad \text{for all } k, l; k < l$$

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). In this model and all subsequent models priority in the branch-and-bound was always given to the γ variables, in decreasing order of crew size. Clearly, the γ variables were the principal variables in the problem, since the x variables were introduced for the sole purpose of modelling the meeting constraints. The motivation for considering them in decreasing order of crew size was to fix the hosts for the larger guest crews first, since these crews are most likely to be split up, and consequently the variables associated with them are the most likely to have fractional values in the LP relaxation. Giving these variables priority would rule out infeasible allocations earlier on in the search process.

The PS1 model gave an optimal solution with 5 hosts in a total time of 2,214 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 (Williams, 1992). 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.

Table 2. Results with MVUB

Model	PS1	PS11	PS12
Rows	4,386	2,130	4,022
Columns	2,271	2,271	2,271
LP solution time (secs)	101	16	19
Number of iterations	1,474	696	497
LP objective value	3.42	3.42	3.42
MVUB time (secs)	*n.a.*	561	697
Branch-&-Bound time (secs)	2,113	1,852	3,789
Number of nodes	287	279	311
IP objective value	5.00	5.00	5.00

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 a capacity constraint

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

the following covering constraints could be derived:

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

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

$$\gamma_{131} + \gamma_{141} + \gamma_{151} \leq 2$$

In total, there would be a vast number of such constraints and so attempting to generate them all automatically did not seem a particularly fruitful approach. To generate only those particular constraints which were violated by a given solution would have involved considerable extra programming, which the importance of the problem did not seem to warrant, although this is a recognized and successful technique (see for example Crowder, Johnson and Padberg (1983)). However, experiments with adding a small number of covering constraints by inspection are reported in section 8.

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

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

for index sets Q, where $|Q|$ is odd. By inspection, many of these were violated by the fractional solution. Although automating the process of inspecting the solution, identifying the appropriate index sets and then generating the corresponding cuts would have been possible, it would have been very time-consuming to implement because of the size of the full problem. Moreover, since each individual LP relaxation was taking so long to solve, the total solution time would still be very long, even if branch-and-cut were finally successful in finding a solution.

To summarize, the experiments with the reduced problem did not indicate any solution strategies for the full problem which did not require the addition of extra constraints. The main difficulty appeared to lie in finding a formulation of the problem which did not have a huge number of constraints, so that considerable computer time was occupied merely in handling such a large matrix.

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. (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.)

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[2] on seven RS/6000 computers at IBM UK Scientific Centre, 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 problem to be tackled was whether there was a solution with the 13 host boats selected as described in section 6. Therefore this was exactly the same problem which had already been tackled without success using ILP. The problem was formulated as a constraint satisfaction problem (CSP) and implemented in ILOG Solver (Puget, 1994), 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 (1980). 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 (Haralick & Elliott, 1980)), with additional constraint propagation to re-establish arc consistency after each assignment.

7.1. CSP Formulation

It would have been possible to use the ILP formulation described earlier, more or less directly, since linear inequalities are an allowable form of constraint in CSPs. The main variables would then have been the γ_{ikt} variables described in section 4.1, with domain $\{0,1\}$. However, using virtually the same formulation does not make use of the fact that in constraint programming *any* relation on the domains of the variables can be a constraint: although this includes linear inequalities, the scope is much wider. In fact, in formulating the problem as a CSP, no attention was paid to the existing ILP formulation, and a much more compact representation was found.

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.

(Note that although in the description of the ILP formulation given earlier, variables representing the choice of host boat were included, these had all been assigned values by the time that the unsuccessful attempt was made to prove that the 13-host problem was infeasible, i.e. the 13 host boats had been selected, as here. Hence, the comparison between constraint programming and integer programming on the 13-host problem is a fair one.)

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:

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

and the meeting constraints are expressed by:

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

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 (1993) 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. Because the search algorithm uses the constraints to prune values from the domains of those variables which have yet to be assigned, the variable which is likely to be hardest to assign is the one 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 slightly 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. For instance, a solution with 14 hosts (i.e. the 13 selected hosts together with the remaining boat with largest spare capacity), 28 guests and 6 time periods was found in 2.3 sec.: as described earlier, it had been difficult to find such a solution from the ILP model. Similarly, a solution with 13 hosts and 29 guests could be found very quickly, but only for 3 time periods. 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 as a subproblem, but the solutions for each time period constrained solutions for later time periods. 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. At this point, it was still believed that the problem had no solution, and it was hoped that the variable domains 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 problem with the 13 selected hosts, 29 guests and 6 time periods, 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. This solution is shown in Table 3.

Table 3. An Optimal Solution

Host	T1	T2	T3	T4	T5	T6
1	34,35	30	29,31	14,25	32,40	20,36
2	31,32,33	17,41	14,20,40	24,29	36,37	15,34
3	14,39	20,22,31	16,23	28,38	19,24,35	26,30,42
4	22,28	25,39	24,38	15,16	23,29,42	14,21,37
5	18,38	27,29,33	22,41	39	16,20	28,31
6	19,21	16,34	25,28	17,33,42	39	22,35
7	23,24	15,38	30,37	18,32,41	14,22	16,33
8	16,17	26,28,32	35,39	19,22	21,38	24,27
9	27,30	19,37	21,42	26,31,34,40	15,28	38,41
10	37,42	14,23,36	19,27	21,30	26,33,41	17,18,25,40
11	20,25,26,29	24,42	15,17,32	27,35	18,30,34	19,23
12	15,41	21,40	18,26,36	20,23	17,27	39
13	36,40	18,35	33,34	37	25,31	29,32

Subsequently, the manual assignments to the variables corresponding to 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!

The heuristic of considering time periods separately has since been rewritten as a genuine variable ordering heuristic, so that the next unassigned variable to be considered is one relating to the earliest time period; in the case of a tie, one with the smallest remaining domain; in the case of a further tie, one involved in the largest number of constraints, and finally, in the order specified (largest crew size first). This allows the program to backtrack to earlier time periods in case of failure. In fact, with hindsight, this heuristic expresses the fail-first principle: the capacity constraints are harder to satisfy than the meeting constraints, so that an assignment of a host boat to a guest boat is more likely to lead to a failure elsewhere in the same time period than in a different time period. It also corresponds to the way in which a person attempting to solve the problem would be likely to tackle it.

8. Modifications to the ILP Formulation

The success of the CSP approach led us to reconsider our ILP formulation and solution strategy. Firstly, we tried to replicate the CSP formulation as far as possible by replacing the x_{ijkt} variables by the m_{jkt} variables, recognizing that the boat on which two guest crews meet is irrelevant. This reduced the number of variables from 33,931 to 4,699, but the number of rows was unchanged at 64,372. The problem lies in linking the m and the γ variables:

$m_{jkt} = 1$ iff $\gamma_{ijt} = 1$ and $\gamma_{ikt} = 1$ for some i

This requires two constraints (similar to the S and V constraints given in section 4.2) for each i, j, k and t combination, giving $2 \times 13 \times \binom{29}{2} \times 6 = 63{,}336$ constraints altogether, which exceeds the limits of the XPRESSMP modeller. Reducing the number of variables is obviously advantageous, but the computational experience with the reduced model (379 variables and 18,212 rows) indicated that this model would still be too large to solve. The CSP formulation is compact because of the "if ... then ... " constraints which link the m and h variables, and the "all different" constraints, which cannot be expressed compactly in ILP form.

Secondly, we reconsidered the problem of symmetry. This was an issue to which we had already given much thought. Eliminating symmetry is clearly desirable, from the nature of the data; for example, there are 14 crews of size 2, any pair of which could be interchanged in a feasible solution to give another feasible solution. The difficulty of symmetry-breaking in the ILP lies in avoiding the addition of a large number of extra constraints to a problem which is already very large. One of the reasons that the symmetry constraints can be neatly expressed in the CSP model is that the variable h_{jt} represents the actual index number of the host boat visited by guest j at time t. Thus for example it is possible to rule out solutions where guests j and k, of equal crew size and with $j < k$, could be interchanged, by adding the constraints:

$$h_{j1} \leq h_{k1}$$

$$\text{either } h_{j1} < h_{k1} \text{ or } h_{j2} < h_{k2}$$

as in section 7.2.

To represent the first set of constraints in the ILP, for example for the 14 crews of size 2, for all pairs of indices j, k, where $c_j = c_k = 2$, and $j < k$, we have:

$$\gamma_{ik1} + \sum_{m=i+1}^{13} \gamma_{mj1} \leq 1 \qquad \text{for } i = 1, ..., 12$$

and there are $12 \times \binom{14}{2} = 1{,}092$ such constraints.

A similar set of constraints could be added for the crews of size 4. Although the addition of extra constraints would increase the solution time of the LP relaxation, the hope was that by eliminating equivalent feasible solutions, the size of the branch-and-bound search tree would be significantly reduced.

Thirdly, armed now with the knowledge that the problem was feasible, we adopted the same approach of solving the problem one period at a time. For the first period we constructed a model, called 1PER, in which the only variables required were γ_{ij}, meaning that boat j was a guest of boat i. Clearly for a single period no "meeting" constraints were required, and there was no time dimension, so the constraints were simply the capacity constraints, a small number of covering constraints (discussed below), the symmetry-breaking constraints described above, and the GA constraints. We experimented with various combinations of symmetry constraints, and the fastest solution time was obtained for the model

using those for the crews of size 4 only: this model had 378 variables and 392 constraints. The LP relaxation was solved in less than one second, and branch-and bound took 18 seconds to obtain an integer solution.

A second model which we called 2PERA was then constructed. Again, this model had no time element, but it was necessary to prevent crews which had already met from meeting again, so we introduced the m_{jk} variables. The values for which $\gamma_{ij} = 1$ in the solution of 1PER were manually fixed at zero, by inspecting the solution, and similarly the values of m_{jk}, where j and k had met in the solution of 1PER were fixed at zero. The constraints were the same as for 1PER, without the symmetry constraints (which applied only to the first period) but with the addition of constraints linking the m and the γ variables:

$$m_{jk} = 1 \text{ iff } \gamma_{ij} = 1 \text{ and } \gamma_{ik} = 1 \qquad \text{for some } i$$

This model had 784 variables and 10,612 constraints. The LP relaxation solved in 31 seconds, but the branch-and-bound was still running overnight.

We then constructed another model for the second time period without using the m variables at all. Hence in addition to fixing some of the γ variables as in 2PERA, by inspecting the solution of 1PER we manually added further constraints to express that if crews j and k met in period 1, then they cannot meet in period 2:

$$\gamma_{ij} + \gamma_{ik} \leq 1 \qquad \text{for } i = 1, ..., 13$$

This model had 378 variables and 355 constraints. An integer solution was obtained in 15 seconds. This solution was inspected and the process repeated for the third time period, giving a model 3PER with 641 constraints, and an integer solution for the third time period was obtained after 6 seconds. However, the model for the fourth period (which had 378 variables and 914 constraints) ran overnight in branch-and bound without finding a solution, despite the fact that the LP relaxation solved in less than 2 seconds.

In the course of the experiments with the single period models, we investigated the effects of adding covering constraints, as described in section 5. The potential number of such constraints is very large, but some useful ones can be derived by inspection. Because of the small capacities, no more than two crews of size ≥ 4 can ever be accommodated simultaneously, giving constraints of the form:

$$\sum_{j, c_j \geq 4} \gamma_{ij} \leq 2 \qquad \text{for } i = 1, ..., 13$$

We ran the 2PERA and 3PER models described earlier with and without these 13 constraints. In the 2PERA model, leaving out these constraints increased the solution time of the LP relaxation from 31 seconds to 250 seconds, but in both cases branch-and-bound failed to find a solution when left to run overnight. Similarly, without these constraints the branch-and-bound solution time for the 3PER model increased from 6 seconds to 325 seconds, although the LP relaxation solution time was still less than 1 second. Thus we included these covering constraints in all the single-period formulations.

Despite some successes, our overall conclusion was that the modified ILP formulation described in this section, which was very close in spirit to the CSP approach, had still failed to find a solution to the six-period problem, even with a high degree of manual intervention.

9. 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. Moreover, for those problems which linear programming succeeded in solving, constraint programming found solutions much more quickly.

It should be stressed that the constraint programming solution required not only far less computer time, but also much less human effort that was expended on the linear programming approach (even ignoring the later experiments described in section 8). Although section 7 necessarily describes the constraint programming approach in some detail, in practice it was very straightforward to formulate the problem, and incorporate symmetry constraints and variable ordering heuristics based on the fail-first principle. It was only the development of the solution strategy based on solving each time period separately that involved any experimentation; even then, the implementation required very little programming effort.

A number of reasons to account for the success of constraint programming in this case can be identified.

9.1. 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 much weaker assumptions about the form 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 search 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. (Note that a CSP formulation using the γ_{ikt} variables of

the ILP as its principal variables would have had $2^{13\times29\times6}$ possible assignments, which is many times larger.)

9.2. *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.

9.3. *Solution Strategy*

In both the ILP and CSP approaches, attempts were made to find solution strategies appropriate to the problem. However, in the ILP approach many of the experiments that were tried, as described in section 5, required adding constraints to the model, and since the ILP formulation was already very large, this was an obstacle to the successful development of good solution strategies. Furthermore, it is not easy to see the original problem in the ILP model: for instance, the majority of the variables are the 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. This makes it more difficult to translate from reasoning about how to solve the problem into solution strategies for the ILP.

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, and not easily generalized, 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 specialized heuristic, but it can be seen as an extension of the fail-first variable ordering heuristic.

9.4. 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.

9.5. 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. 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. As an example of this, no solution has been found with the 13 specified host boats for 8 time periods, but nor has it been shown that there is no solution.

10. Related Work

Dincbas, Simonis and van Hentenryck (1988) and van Hentenryck and Carillon (1988) also compare integer linear programming and constraint programming applied to particular problems. The latter paper describes a warehouse location problem, and suggests 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 9.3. 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.

Dincbas, Simonis and van Hentenryck (1988) discuss 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. In the paper cited, 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 CSP formulation still has a relatively small number of values compared with the number of variables, and the number of possible assignments is $13^{29\times 6}$. 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.

Puget and De Backer (1995) compare 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 9.2, 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 (1995) discuss the combination of the two approaches for optimization 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.

11. 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 9. Some of these reasons have been identified in other studies of problems where constraint programming out-performed ILP, as discussed in section 10. 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 than 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 recognized 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.

Acknowledgments

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.

Notes

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

2. Optimization Subroutine Library (OSL), IBM Corporation.

References

Beringer, H. & De Backer, B. (1995). 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..

Crowder, H., Johnson, E.L. & Padberg, M.W. (1983). Solving large-scale zero-one linear programming problems . *Operations Research*, 31:803–834.

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

Haralick, R.M. & Elliott, G.L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313.

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

Puget, J.-F. (994). A C++ Implementation of CLP. In *Proceedings of SPICIS94 (Singapore International Conference on Intelligent Systems)*.

Puget, J.-F. & De Backer, B. (1995). Comparing Constraint Programming and MILP. In *APMOD'95*, 1995.

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

Williams, H.P. (1992). The elimination of integer variables. *JORS*, 43:387–393.