Constraint Logic Programming and Integer Programming approaches and their collaboration in solving an assignment scheduling problem

Ken Darby-Dowman (1)

James Little (1)

Gautam Mitra (1)

Marco Zaffalon (2)

(1) Department of Mathematics and Statistics, Brunel University, Uxbridge, Middlesex, UK, UB8 3PH

(2) Università degli Studi di Milano Dip. di Matematica, Via Cicognara 7 20129, Milan, Italy

7th September 1996

An earlier version of this paper was first presented at the Third International Conference on Applied Mathematical Programming and Modelling (APMOD'95), held at Brunel University, 3-5 April, 1995.

Abstract

Generalised Assignment Problems (GAP), traditionally solved by Integer Programming techniques, are addressed in the light of current Constraint Programming methods. A scheduling application from manufacturing, based on a modified GAP, is used to examine the performance of each technique under a variety of problem characteristics. Experimental evidence showed that, for a set of assignment problems, Constraint Logic Programming (CLP) performed consistently better than Integer Programming (IP). Analysis of the CLP and IP processes identified ways in which the search was effective. The insight gained from the analysis led to an Integer Programming approach with significantly improved performance. Finally, the issue of collaboration between the two contrasting approaches is examined with respect to ways in which the solvers can be combined in an effective manner.

Keywords

Constraint Logic Programming, Integer Programming, Generalised Assignment Problem, Optimisation

1. Introduction

The analysis of complex decision problems, once the preserve of Operations Research (OR), has in recent years attracted the attention of analysts from other disciplines. In particular, Artificial Intelligence (AI) in the form of Constraint Logic Programming (CLP) is now tackling serious decision problems in the areas of scheduling [Duncan (1994)], resource allocation [Dincbas and Simonis (1991)] and planning [Bellone et al. (1992)]. These applications generally involve some form of combinatorial search and present challenges in both modelling and developing efficient search strategies. Combinatorial optimisation problems are often NP-hard and, as such, all known exact algorithms have an exponential worst case complexity. This is not to say that such problems are necessarily unsolvable even when large scale. However, most approaches to solving large scale combinatorial optimisation problems use, at some stage in the solution process, a tree search. The key to a successful approach is to adopt a ‘good’ branching strategy whereby the tree search can be completed relatively quickly.

Integer programming (IP) and Constraint Logic Programming (CLP) are two alternative approaches to modelling and solving combinatorial optimisation problems. Integer Programming is an established approach within Operations Research and modern

software allows substantial user control over the branching strategy. Essentially the search is guided by factors such as the mathematical structure of the integer polytope, the extent of the fractionality of the current solution or the priorities allocated externally to various subsets of variables. Constraint Logic Programming has an important role within the Artificial Intelligence approach to problem solving. Here again, the key to success lies in directing the search whereby the analyst can exploit problem specific features in determining search strategies, complementing the underlying constraint solvers' own search techniques. A full description of the CLP paradigm is presented in Little and Darby-Dowman (1995).

During the evolution of CLP, several authors have compared it to the technique of IP. An early example is presented by Van Hentenryck and Carillon (1988) who compared CLP and IP applied to a simple warehouse location problem. The comparison was based mainly on their different modelling capabilities. As CLP has developed, the size of problem capable of being tackled has grown. Smith et al. (1995), El-Sakkout (1995) and Hajian et al. (1995) have all tackled large-scale problems with 'state of the art' solvers and high degrees of expertise. The first paper describes a problem of finding a constrained set of permutations of 'guests' visiting 'hosts' over several discrete time periods. A property of this particular problem is that any feasible solution is necessarily optimal. Thus the search can be terminated as soon as a feasible solution is obtained irrespective of whether or not the search tree has been fully fathomed. The conclusion was that Constraint Programming gave better solutions than Integer Programming due to the compactness of the problem representation and the effectiveness of the constraints in reducing the search space. The solutions required manual amendment, but only after CP had provided a good starting point. Both solvers were used with a high degree of sophistication. The second and third papers are both concerned with the assignment of aircraft to flights under a set of operational constraints. El-Sakkout showed that an optimal solution was obtained only from IP, but a good (sub-optimal) solution was produced quickly by CLP. An analysis of why the approaches differed in terms of performance was not fully explored. However, Hajian et al. observed that, for the problems they considered, the CLP approach was able to produce a feasible solution quickly whereas IP was good at proving optimality. By combining both approaches, a 'loose-coupled' hybrid solver was able to achieve improved performance over that of each individual approach.

Due to the high levels of expertise needed for both solvers, research in the area of solver comparison has been relatively unexplored to date. It is only by understanding the reasons for solver performance on problems can the right choice or combination be made in the future. In this paper, an analysis is made of the solving performance of IP and CLP for a type of assignment problem, using a number of different strategies. The problems are solved to optimality and require a full search of the solution space. The problem is taken from a real manufacturing assignment problem arising in the Italian telecommunications industry.

It is well known in the evaluation of optimisation algorithms that computational behaviour on randomly generated problems may not be indicative of performance on real problems [Crowder et al. (1978)]. Nevertheless, much experimental work uses randomly generated problem sets. The advantage is that problem characteristics can be fully controlled and larger problem sets can easily be produced. Experiments with using real datasets are restricted by the availability of relatively few problems to analyse, but the advantage is that real life behaviour is being observed rather than predicted. In this study, there are four real datasets with which to evaluate the two solving techniques.

The paper is organised as follows. Section 2 describes the problem in terms of its real world context. Section 3 describes the IP formulation as a Modified Generalised Assignment Problem (MGAP) problem and compares it with the corresponding Generalised Assignment Problem (GAP) formulation. Section 4 describes the CLP formulation of the problem. Section 5 describes in detail the investigation into the comparison between the two techniques. The characteristics of the problems used in the computational experiment and the results are also presented. The interpretation of these results is given in section 6. Concluding remarks and discussion are presented in sections 7 and 8 respectively.

2. Introduction to the Problem

The experimental work reported in this paper is based on a sub-problem arising in the manufacture of telecommunications cabinets. The overall application is first described and then the sub-problem, an allocation of manufacturing tasks to machines, is presented.

2.1 Overall Context of the Application

The problem addressed here is one where there is a requirement to produce specified numbers of different types of telecommunications cabinets over a fixed time period. Each cabinet type requires a different set of elementary sequential operations, each taking the same time to be carried out. The manufacturing process involves a number of identical unit cells linked together to form a pipeline machine capable of carrying out the tasks. Several machines make up a configuration. Figure 1 shows an example of a 17 cell, 11 machine configuration.

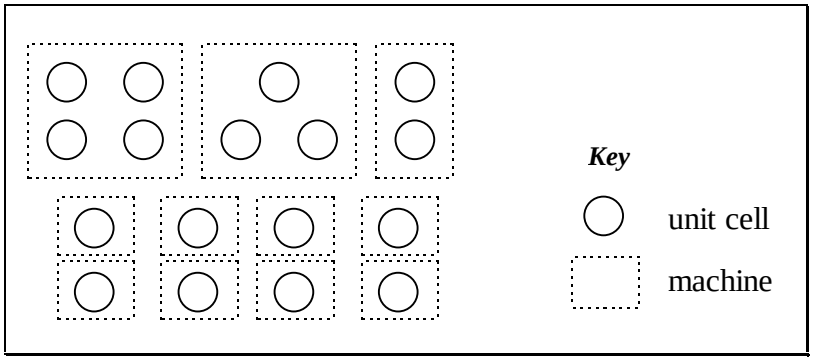

Figure 1: An example of a machine/cell configuration

Each unit cell is capable of executing all operations to build any particular type of cabinet. A machine with a particular number of cells is allowed only to manufacture cabinets where the number of operations is a multiple of the number of cells. For example, a five cell machine will only be suitable for manufacturing cabinets comprising of 5, 10, 15 ... operations, whereas a two-cell machine is suitable for building cabinet types with 2, 4, 6... operations. The first stage of the problem is to select the set of machine/cell configurations to be used.

2.2 The Assignment Problem

In this case study, it is assumed that the selection of machine/cell configurations has already taken place and the set of requirements for each cabinet type is known. All cabinets of a certain type must be manufactured on the same machine. A task is therefore defined as the work required to manufacture a set of identical cabinets. The problem is to allocate each task to a machine such that the production across all machines is completed as early as possible, i.e. minimise the longest makespan.

2.3 An Example of the Problem

Each instance of the assignment problem starts with the data specifying a particular configuration of machines to be used and a set of tasks to be done. Tables 1, 2 and 3 below show example cabinet and machine data for a small problem.

Type of cabinet	Number of cabinets required	Number of operations	Time to produce one cabinet on a single cell (hours)	Manufacturing Restrictions $\{r_{ij}\}$			
1	4	4	7.8	0	1	1	0
2	4	3	5.1	1	0	0	1
3	5	2	3.2	1	0	0	0
4	1	2	2	0	1	0	0

where $r_{ij} = 1$ if cabinets of types i and j cannot be manufactured on the same machine and $r_{ij} = 0$ otherwise.

Table 1: Example cabinet data

n-cell machine n	Number of machines available	Capacity of each machine (hours)
1	8	50
2	2	50
3	1	50
4	1	50

Table 2: Example machine data

Task	n-cell machine			
	n = 1	2	3	4
1	31.2	19.5	x	13.65
2	20.4	x	10.2	x
3	16	9.6	x	x
4	2	2	x	x

Table 3: Durations (hours) of tasks on different n-cell machines

In Table 1, the fourth column gives the time for manufacturing one cabinet on a single cell. These values are used to calculate the time that each machine requires to manufacture cabinets of each type (see Table 3), according to a known pipeline workstation formula (provided that the number of operations is divisible by the number of cells). The final column contains a Boolean square matrix that specifies which cabinet types cannot be produced on the same machine. For instance, the two 1's in the first row indicate that cabinet types 2 and 3 cannot be produced on the same machine as cabinet type 1.

The last column of Table 2 shows the maximum number of machine hours that are available for each n-cell machine type, which in this application, is the same for all machines. The solution to the problem specifies the machine to which each task is assigned and the optimal makespan.

3. Integer Programming Models

3.1. Integer Programming Model for GAP

The Generalised Assignment Problem (GAP) has been proposed for many practical applications in business and industry. Examples include facility location [Ross and Soland (1977)], vehicle routing [Fisher and Jaikumar (1981)] and production planning [Shtub (1989)].

The Generalised Assignment Problem determines the least cost assignment of n tasks to m machines where c_{ij} represents the cost of assigning task j to machine i and t_{ij} is the amount of machine resource required when task j is assigned to machine i. The total availability of resource for machine i is b_i.

The IP formulation of GAP is shown below.

$$\text{minimise } w = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

subject to

$$\sum_{j=1}^{n} t_{ij} x_{ij} \le b_i \quad i \in M = \{1,...,m\},$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad j \in N = \{1,...,n\},$$

$$x_{ij} = 0 \text{ or } 1 \quad i \in M, j \in N$$

where

$$x_{ij} = \begin{cases} 1 & \text{if task j is assigned to machine i, } i \in M, j \in N \\ 0 & \text{otherwise.} \end{cases}$$

The assignment problem contained in this paper is similar to GAP and a modified model, known as the Modified Generalised Assignment Problem (MGAP) is described. Stella et al. (1994) provide a fuller description of the overall application and IP formulation.

3.2 Integer Programming Model for MGAP

The mathematical model for the Modified Generalised Assignment Problem (MGAP) is presented below.

Let M = {1,...,m} be the set of machines in the given configuration.

Let N = {1,...,n} be the set of tasks corresponding to cabinet types required to be produced.

Let F_j be the set of the machines capable of processing task j ($F_j \neq \varnothing$).

Let $S_{j_1 j_2} = \begin{cases} 1 & \text{if tasks } j_1 \text{ and } j_2 \text{ are not allowed to be processed on the same machine} \\ 0 & \text{otherwise} \end{cases} \quad \forall (j_1, j_2), \quad j_1, j_2 \in N .$

Let $S = \left\{ (j_1, j_2) \middle| j_1 < j_2, S_{j_1 j_2} = 1 \right\}$ be the set of pairs of incompatible tasks.

Let t_{ij} be the time required for task j to be processed on machine i.

Let b be the total time available for each machine.

Let X= $[X_{ij}]$ be the Boolean matrix of the decision variables, where $X_{ij} = 1$ means that task j is assigned to machine i.

Let W represent the limit on the makespan for each machine and since W is to be minimised, its value at optimality will equal the longest makespan over all machines.

MGAP can therefore be expressed as,

Min W

subject to

(1) $\sum_{j \mid i \in F_j} t_{ij} X_{ij} \leq W , \quad \forall i \in M$

(2) $\sum_{i \in F_j} X_{ij} = 1 , \quad \forall j \in N$

(3) $W \leq b$

(4) $X_{ij_1} + X_{ij_2} \leq 1 \quad \forall i, \forall (j_1, j_2) \in S$

$W \geq 0 , \; X_{ij} \in \{0,1\}, \; \forall i \in F_j , \; \forall j \in N$

The four sets of constraints may be interpreted as follows.

(1) The makespan on each machine must not exceed W. By minimising W, an assignment that minimises the maximum makespan is obtained;

(2) every task must be assigned to exactly one machine;

(3) a makespan limit of b is imposed. An allocation in which the maximum makespan exceeds b is therefore infeasible; and

(4) at most only one of tasks j_1 and j_2 can be allocated to any machine.

The major differences between GAP and this modified version MGAP, relate to
(i) the extra constraints in MGAP, representing assignment restrictions (4),
(ii) the insertion of the makespan variable W, in the capacity constraints (1) and (3),
(iii) the different cost function.
In a GAP model, each cabinet type can be assigned to any machine, whereas in the MGAP model, certain pairs of cabinet types cannot be assigned to the same machine. The capacities in GAP are known constants, whereas in MGAP, they are represented by the makespan variable W. The objective function in GAP is to minimise the total cost of allocating tasks to machines, whereas with MGAP, the objective is to minimise the maximum makespan.
GAP is known to be NP-hard [Martello and Toth (1990)] and most success in solving this type of problem has been found using specialised algorithms. Problems that are especially hard to solve are those with tight bounds on the capacity constraints [Martello and Toth (1981)]. This property is present in the application considered here.

4. Constraint Logic Programming Model

The CLP approach to the problem differs from that of the IP approach for two related reasons: the underlying solver and the availability of certain high level constraints. In CLP, the solver operates by creating a search tree based on enumeration. The problem is then modelled in a way which will allow the maximum amount of search reduction through activation of the constraints during the search process. Secondly, the use of certain constraint types, such as the element constraint, can be exploited by the solver to achieve search reduction.
Since logic programming is both a modelling and solving language, the model for this problem will be described in a general logic syntax. There is currently no standard for describing CLP problems. A number of ways of representation appear in the literature.

These ways include presenting the model: as a complete CLP program, in algebraic form similar to IP models and as a combination of algebraic nomenclature within a CLP program style. The syntax adopted here is the third style and is similar to that used by Van Hentenryck and Carillon (1988). To aid comparison of the two approaches, there will be cross-referencing between the IP constraints and those in the CLP model.

4.1 Defining the Task Assignment Variables

The model is created using a set of finite domain decision variables, each one representing a task whose value represents the machine to which it is assigned. Therefore, given n tasks (1,...,n) and m machines (1,...,m), let X_j represent the machine to which task j is assigned.

Each task is to be assigned to a single machine and this requirement is represented as

$[X_1, X_2, ... ,X_n] :: 1.. m$

which states that each of the variables $X_1, X_2, ... ,X_n$ must take one of the values 1,2 ...,m. This constraint corresponds to constraint set (2) of the IP model.

4.2 Excluding Tasks from the same Machine

Applied to these decision variables $\{X_1, X_2, ... ,X_n\}$ are the constraints relating to the disjoint machine allocation, in which specified pairs of tasks cannot be carried out on the same machine. Therefore, if task j and task k cannot be assigned to the same machine, this is represented as

$X_j \backslash= X_k$

where $j \neq k$ and $1 \leq j, k \leq n$.

Similar statements are made for each disjoint combination of tasks. This set corresponds to the IP constraint set (4).

4.3 Relating the Choice of Machine to the Duration of the Operation

Since for each machine there is the same fixed number of hours available (b in the IP model), a constraint must ensure that this is never exceeded. In addition, there is a known duration that a task takes when allocated to each available machine. Therefore, for each machine i available to task j, there is a constraint of the form

element(X_j ,[0, ..., t_{ij}, ...,0], D_{ij})

where t_{ij} is the duration of task j on machine i and appears in the i'th position in the argument list.

The decision variable D_{ij} has a value corresponding to the X_j th member of the list and is either t_{ij} or 0 depending on whether task j is assigned to machine i or not

i.e. whether $X_j = i$ or $X_j \neq i$.

These constraints are imposed for every available machine and task combination.

The capacity constraint for machine i is therefore expressed as,

$ms_i = D_{i1} + D_{i2} + ... + D_{in}$

$ms_i \leq b$

where ms_i is the makespan for machine i and b is the time limit on each machine. The set of these constraints for all machines corresponds to the IP constraint sets (1) and (3).

4.4 Defining the Cost Function

The cost function to minimise the longest makespan is expressed in CLP using the optimisation function *min_max* as,

$$min\_max(\ search([X_1, X_2, ... ,X_n]), [ms_1, ... ,ms_m])$$

where the *search* function gives values to the task assignment variables $X_1, X_2, ... ,X_n$ in such a way that they satisfy all the constraints and minimise the maximum value within the set $\{ms_1, ... ,ms_m\}$. This corresponds to the IP objective function and to constraint set (2).

The speed at which feasible solutions and subsequently the optimal one are found, depends on the strategy adopted in searching. Within CLP, the constrained search is made over a tree formed by enumeration of all variables. Therefore, the order of choosing the variables and the order in which values are assigned to these variables forms the strategy.

4.5 Summary of the Model

start :-

$[X_1, X_2, ... ,X_n] :: 1..m,$

$X_j \backslash= X_k , \quad \forall (j,k) \in S$

. . . . . .

element(X_j,[0, ..., t_{ij}, ...,0], D_{ij}), $\quad \forall i \in M, \forall j \in N$

. . . . . .

$ms_i = D_{i1} + D_{i2} + ... + D_{in} , \quad \forall i \in M$

. . . . . .

$ms_i \leq b , \quad \forall i \in M$

min_max(search([$X_1, X_2, ... ,X_n$]), [$ms_1, ... ,ms_m$]).

The CLP model is in some ways closer to the problem owner's statement of the problem. For example, the problem owner is likely to view the problem as one of deciding which machine each task should be allocated, rather than decisions on whether or not each task should be allocated to each machine. The CLP model uses variables that explicitly represent the former set of decisions, whereas IP uses variables that represent the latter. Another example of the similarity between the CLP approach and the problem owner's view of the problem is that the CLP solution process works entirely with possible machine assignments. The IP process, on the other hand, relaxes the constraints and treats them as continuous mathematical statements which are dealt with using real arithmetic and therefore have no real-life interpretation. This similarity between the CLP approach and the user's approach, results in an enhanced ability to use application specific knowledge to develop good search techniques.

The directness of the CLP approach results in only n assignment variables compared to the (m * n) variables of the IP approach. However, the compactness achieved by this aspect of the model is offset by the need to use the (m * n) decision variables D_{ij} and the associated (m * n) constraints that are required in order to define the makespan constraints. These constraints are the *element* ones, available in ECLiPSe. Other CLP solvers can provide higher level constraints which would result in a more compact representation. Thus size is not a meaningful concept for a CLP model, since it depends on the constraint types available within the chosen CLP solver.

The use of global constraints such as the element constraint, leads to a more compact problem description as they replace with one statement, several primitive constraint statements describing the same restrictions. In addition, global constraints can be more effective than the sum of the individual primitive ones. The reason for this is two-fold: the first is that more effective search reduction can be done by considering all the constraints together; the second reason is that the constraint propagation and search reduction facilities are all coded at a lower level than the equivalent of using several primitive constraints at a higher level. This leads to potentially faster execution times.

5. The Comparison of the Two Approaches

The widespread availability of IP solvers has provided many solved instances of combinatorial optimisation problems such as MGAP. The problem considered here is also

one for which CLP would appear to be an appropriate tool. In this paper, the aim is to examine the performance of one CLP solver relative to the better performance of two commercial IP solvers on a set of four MGAP problems. As part of the evaluation, reasons for performance differences are sought in order to further understand why particular characteristics are important for a particular solver.

The proficiency of the user operating both types of solver can be demonstrated in many aspects of the solution process such as the identification of problem features, the way the constraints are expressed, the selection of solver options and the development of the search tree. The purpose of the experiment is not to develop special purpose algorithms for the particular application, but rather to use existing software in the form of commercially available general purpose systems in ways that require no more expertise than typical users of such systems would possess. Given this aspect of the experiment, a comparison can be made at a consistent and realistic level, enabling a meaningful analysis to be made.

The commercial IP solvers used were CPLEX (Version 4.0) [CPLEX OPTIMISATION (1995)] and FortMP (Version 1.04) [Numerical Algorithms Group (1995)]. CPLEX is a widely used commercial IP solver which is acknowledged as being among the leading products of its type in terms of performance. FortMP is a commercial solver which allows considerable flexibility in defining search strategies. Both these solvers have advanced solving features such as pre-processing and heuristics to find initial solutions. ECLiPSe [European Computer Research Centre (1994)] was the system used in the CLP approach. This system has several built-in search strategies as well as the flexibility to define others easily. It also has a number of higher level constraints.

All runs were carried out on a DEC ALPHA 3000/600 with 92Mb of memory.
Four real problems were considered in the experiment. The problems differed in the number of tasks and machines (which affects the size of the problems) and the task durations (which affect the tightness of the makespan constraints). Data for the four problems are presented in Tables 4, 5, 6, 7 and 8.

	Problem 1	Problem 2	Problem 3	Problem 4
No of Machines	3	9	7	8
No of Tasks	8	14	14	14
Makespan limit (hours)	200	56	56	56
No of disjoint machine constraints	0	8	11	13

Table 4: Characteristics of the four assignment problems

Task	Task durations (hours)	
	(Machine Type, Number of Machines Available)	
	(3-cell, 2)	(1-cell, 1)
1	58.67	165
2	32.27	88
3	19.07	55
4	18.24	45.6
5	58.67	165
6	32.27	88
7	19.07	55
8	18.24	45.6

Table 5: Task Durations of Problem 1

Task	Task durations (hours)		
	(Machine Type, Number of Machines Available)		
	(4-cell, 1)	(2-cell, 5)	(1-cell, 3)
1	11.7	19.8	36
2	31.2	54.6	101.4
3	8.8	13.2	22
4	6.75	10.5	18
5	8.12	11.6	16.56
6	7.2	10.8	18
7	7.2	10.8	18
8	32.18	54.45	99
9	10.73	18.15	33
10	13.33	22.55	41
11	14.85	26.4	49.5
12	22.5	40	75
13	51.98	100.65	198
14	25.2	46.8	90

Table 6: Task Durations of Problem 2

Task	Task durations (hours) (Machine Type, Number of Machines Available)		
	(4-cell, 2)	(2-cell, 4)	(1-cell, 1)
1	16.1	29.9	57.5
2	54.6	100.8	193.2
3	13.6	23.8	44.2
4	38.5	70	133
5	9.5	17	32
6	6.65	11.9	22.4
7	18.9	33.6	63
8	21	39	75
9	5.23	8.8	16.5
10	9.45	16.8	31.5
11	30	55	105
12	6	10	18
13	7.5	10	15
14	11.25	20.7	39.6

Table 7: Task Durations of Problem 3

Task	Duration (hours) Tasks (Machine Type, Number of Machines Available)		
	(4-cell, 2)	(2-cell, 3)	(1-cell, 3)
1	11.5	21	40
2	52	96	184
3	9.75	16.5	30
4	38.5	70	133
5	9.5	17	32
6	4.5	8	15
7	18	32	60
8	21	39	75
9	4.5	8	15
10	9	16	30
11	30	55	105
12	6	10	18
13	7.5	10	15
14	11.5	21	40

Table 8: Task Durations of Problem 4

IP solvers allow the user an amount of choice in selecting strategies to control the development of the Branch and Bound tree. These choices are in the form of rules which decide upon the node to develop and the fractional variable at that node upon which to branch. These rules are based on the numerical properties of the fractional solution (e.g. choose the variable which is nearest to an integer value) and do not explicitly take into account application specific knowledge. On the other hand, CLP does allow the problem owner to use problem specific knowledge, both in the modelling and solving processes. In this experiment, two strategies were used in the IP approach:

IP1 : The default strategies were used for both solvers. CPLEX uses a pseudo-cost or maximum infeasibility criterion for choosing the variable to branch on. The choice of branching direction is based on the magnitude of the integer infeasibility of the selected variables. FortMP by default, selects the variable which has value closest to an integer

and branches in the direction of the node which assigns the variable to that closest integer value.

IP2 : The ‘Priority-up’ strategy chooses the first branching direction at any node as ‘up’, towards the value 1 rather than ‘down’ to 0. On backtracking the other branching direction will be taken. This strategy is considered good for many 0/1 IP problems.

Three strategies were used in the CLP approach. They were considered in advance as being ‘sensible’ ones for this type of problem.

CLP1 : The default search strategy was used with task variables being chosen in the order established in tables 5, 6, 7 & 8.

CLP2 : The task assignment variables were chosen using the first-fail search strategy [Haralick and Elliott (1980)]. This strategy dynamically selects the next task to assign to a machine as the one with the least number of possible machines on which it could be allocated. This strategy is already built into ECLiPSe as the *delete* function.

CLP3 : The task assignment variables were chosen in the order of largest work content, based on the duration of the task on a single cell machine.

In addition to these CLP strategies, there are certain symmetry constraints which were added when two or more tasks were equivalent. For example, if tasks i and j had equivalent work content and could go on the same machines, then an ordering could be imposed on them without removing any solutions i.e. $X_i \leq X_j$. Also, the assignment of a machine to each task is by default made by order of the largest machine first, which is a recognised good heuristic for assignment problems.

The results in Table 9 show the performance characteristics of the two approaches on the set of four problems. The results quoted for the IP approach are those obtained by CPLEX which performed better than FortMP on the four problems considered.

		Problem 1	Problem 2	Problem 3	Problem 4
LP solution value		107.1	50.71	54	50.25
First Feasible Solution (obtained by strategy IP1)	Value	123	54.6	55.8	55.0
	Nodes processed	3	37	103	58
	Time (sec)	0.0	0.12	0.4	0.15
Optimal IP	Value	110	54.6	55.8	55.0
Strategy IP1	Nodes processed	272	225 126	96 522	65
	Time (sec)	0.23	401.9	251.3	0.17
Strategy IP2	Nodes processed	269	156 946	93 385	37
	Time (sec)	0.22	270.9	240.2	0.12
First Feasible Solution (obtained by strategy CLP1)	Value	181.88	55.1	55.8	55.0
	Time (sec)	0.05	0.2	0.13	0.15
Optimal CLP	Value	110	54.6	55.8	55
Strategy CLP1	Time (sec)	0.2	0.3	31.7	0.33
Strategy CLP2	Time (sec)	0.27	0.3	11.7	0.2
Strategy CLP3	Time (sec)	0.17	0.22	10.0	0.27

Table 9: Characteristics of the IP and CLP solvers on the four MGAP problems

6. Interpretation of Results

6.1 Interpretation of Integer Programming Results

Although there is little to choose between the two IP strategies, strategy IP2 (Priority-up) gave consistently better execution time performance. Optimality was proved in all cases, but within two quite different time bands. The reason for problems 1 and 4 being solved quickly was as a consequence of the size of their search space. In the case of problem 4 this size was achieved only after CPLEX pre-processing had reduced the initial problem size by eliminating 37 out of 127 rows and 21 out of 112 columns. Pre-processing was unable to achieve the same effect for problems 2 and 3. In these two cases, the first feasible solution was found quickly and, although this was the optimal value, the search to prove that it was optimal took the majority of the time. It was necessary to examine a relatively large number of nodes, with potential for yielding an improved solution, before discovering that a distinct improvement could not be achieved. In these problems, there are many equally optimal solutions since it is possible to swap many pairs of tasks without altering the minimum makespan. Consequently, many paths of the search tree cannot be fathomed early in the tree.

6.2 Interpretation of Constraint Logic Programming Results

All three strategies for CLP performed well on all four problems with execution times ranging from 0.2 to 31.7 seconds to find optimality.

The performance was robust, but differences within a particular strategy were due not only to the size of problem (problem 1 being much smaller) but also to the degree to which effective constraint propagation took place. This latter reason concerns the numbers of constraints and their tightness in being able to eliminate large parts of the search space as early as possible. The choice of search strategy had least effect on problems 1, 2 and 4, all of which were solved relatively quickly. Strategies CLP2 and CLP3 were better than CLP1 since fewer backtracking steps were required in proving optimality. This is due to an early choice of good assignments with the result that further search to detect eventual failure was not required.

The overall solution process may be described as follows.

(1) The task assignment variables were created and the three sets of constraints (disjunctive machine constraints, timings constraints and makespan constraints) were

applied. At the end of this stage, the domains of the task variables had been reduced for two reasons:

i) Makespan constraints eliminated certain task to machine combinations since their durations exceeded the initial makespan limit on that machine

and

ii) if the previous domain reduction resulted in the domain of a variable becoming a single value, then the appropriate disjunctive constraint was woken. The value from the domain of the other variable in the disjunctive constraint was then automatically removed, ensuring consistency.

(2) Search for the first feasible solution.

The search commenced by choosing task assignment variables according to the given strategy. During the search, domain reduction took place due to constraints being activated for the following reasons:

i) the appropriate disjunctive constraint was woken up when one of its task variables was assigned a value. To keep this constraint consistent, the assigned value was taken out of the domain of the other task variable, thereby preventing it from being assigned to the same machine

and

ii) the appropriate element constraint was woken up whenever a task variable was assigned a value. This meant that one machine now had less available processing time and so several other tasks which required more than the time available on that machine could be removed from the search. This is reflected in the machine values being automatically deleted from the domains of the task variables.

(3) Continued search for optimality.

Once all the variables had been assigned for the first time and a cost had been obtained, a new cost constraint was introduced and the search automatically restarted from the beginning. This new constraint stated that the cost should be strictly less than the best so far. At this point, certain tasks can be logically excluded from being on certain machines. The difference in times between the problem set {2 & 4} and {3} is a direct result of the durations of some tasks lying between the first feasible cost and the makespan limit. In particular, when both problems 2 and 4 have reached a first feasible solution of 54.6 and 55 respectively, then the constraint of makespan < 54.6 or makespan < 55 was

automatically put on and the search continued. Consequently, if there were tasks whose durations lay between the new makespan limit and the original limit (56) on certain machines, then the domains of the task variables were reduced to exclude these. In the case of problem 2, this restricted one of the task assignment variables to a single machine with a duration equal to the new best cost. Therefore, further searching was stopped and optimality was proved. In the case of problem 4, this imposed constraint restricted 3 tasks to sharing 2 machines and since the sum of any of their durations was greater than 55, further searching was stopped and optimality was proved. For problem 3, there was no search reduction possible after the first feasible cost was found.
All problems took around the same time to find the first solution, only in the case of problem 3 was the time between first and optimal solution of any significance.

6.3 Further Investigation of the CLP and IP Approaches

Since CLP appeared to be more successful in solving this type of problem consistently in reasonable time, consideration was initially made as to how CLP could improve the performance of the IP solvers. The initial domain reductions available within CLP can act as a pre-processor to IP, but was found to make no difference to the times for solving the IP models. This is not surprising since IP pre-processors adopt similar logical constraint reductions at the root node. However, unlike CLP, commercial IP solvers do not continue to do this at each node in the search tree.

To illustrate the scope for applying pre-processing within the IP search tree, consider two sub-problems based on problem 3, created by CLP and solved using IP. These sub-problems were obtained by selecting the task assignment variable with the smallest domain after all the initial constraints had been put on. This variable had two possible values: each was chosen in turn and constraint propagation took place resulting in two smaller problems (3a and 3b) which were each passed to the IP solver. The solution times and number of nodes investigated for both sub-problems and the full problem are shown in Table 10. The total time for the two sub-problems was over 40% less than the time required to solve the full problem, with a 23% reduction in the corresponding number of nodes investigated.

	Sub-problem			
	3a	3b	Total	Problem 3
Time to optimal IP (sec)	73.5	74.5	148	251.3
Number of nodes	37007	37700	74707	96522

Table 10: Effect of decomposition on Problem 3

Based on this experiment, one may suppose that, if the domain reductions available from CLP could be integrated further with IP, then improved reductions would be achieved. This point was further illustrated on problem 2 where, once again, a two-stage process was adopted. Firstly, an IP run was made to obtain the first integer feasible solution. At this point, the makespan constraint was tightened based on the cost of the first solution. IP with initial pre-processing was run until optimality was obtained and proved. This process was equivalent to carrying out a single IP run with intra-processing applied at the node corresponding to the first integer feasible solution. The result was that optimality was established after a total of 213.0 seconds and 117,219 nodes compared to 270.9 seconds and 156,946 nodes for the original run.

7. Conclusions

In this paper, Integer Programming and Constraint Logic Programming were applied to a set of four real manufacturing assignment problems. The purpose was to gain insight into the behaviour of both approaches and to examine ways in which the two approaches might collaborate. The results showed that CLP performed well compared to IP which gave a variable and unpredictable performance. For both techniques, performance is governed by the ability to reduce the search space. In Integer Programming, a very large number of nodes were often needed to be considered before the tree was fathomed. The effectiveness of CLP's search space reduction is achieved by performing what, in IP terms, would be a form of intra-processing during the tree search. The experiments reported here suggest that a hybrid system based on CLP putting on local cuts could prove effective for IP in solving problems of this type. However, the effectiveness shown here for improving IP was still inferior to the performance of CLP.

The searching processes within IP and CLP are similar for these model instances. The IP search proceeds by creating branches according to whether a task is assigned to a particular machine or not ($X_{ij} = 1$ or $X_{ij} = 0$). In the CLP case, the branches are created according to which machine a task is assigned ($X_i = j$).

It follows that the assignments made during CLP are equivalent to those made by IP according to the following conversions,

IP: $X_{ij} = 0 \equiv$ CLP: $X_i \neq j$

IP: $X_{ij} = 1 \equiv$ CLP: $X_i = j$

Constraint propagation within CLP can give rise to certain domain reductions or equivalently logical cuts within IP. Therefore, co-operation is achievable at quite a low level from CLP to IP since their models and solving processes have been shown to be similar.

One benefit of modelling a problem in CLP is the insight it gives to the problem structure as opposed to the model structure. The experiments reported in this paper show that this insight can be used to good effect in devising efficient search strategies.

8. Discussion

This paper has shown that for a particular type of problem, CLP outperforms IP. Many problems exist for which it is likely that IP will outperform CLP even within the general class of assignment problems. Furthermore, there are other techniques which address the same type of combinatorial problems. It is therefore desirable to have a set of test problems for which different solvers can be benchmarked. Existing problem libraries such as MIPLIB [MIPLIB (1996)] for IP solvers, are however unsuitable for other solvers. The reason for this is that the problems are presented as an IP formulation and as a consequence, much of the problem structure has been lost. Other libraries contain generated problems or real problems without a full description and are therefore inappropriate for evaluating different technologies.

In order to compare different solving technologies effectively, a library of test problems is needed in which a full problem description together with the associated data is provided. The description should include any heuristics or strategy the manual solver may employ. Such test problems can then be modelled in an appropriate manner for each technique. Two good examples of public domain problems are, the BT Workforce Scheduling Problem [British Telecom Laboratories] and the Radio Link Frequency Assignment Problem [City University Constraints Archive].

The data files associated with the 4 problems in this paper are available with anonymous ftp from ftp.brunel.ac.uk/maths/cplib/cabinet[1,2,3,4].

Acknowledgements

The work on the case study was partly funded through the Human Capital Mobility Programme, Contract CHRX-CT93-0087.

References

Bellone. J, Chamard. A and Pradelles. C (1992), 'Plane': An Evolutive Planning System for Aircraft Production Written in CHIP, *Proceedings of the Practical Applications of Prolog Conference*, London, England, April 1992.

British Telecom Laboratories, *250-118 data sets of Workforce Management Scheduling Problem*. With permission of British Telecom, "http://cswww.essex.ac.uk/CSP/wfs".

City University Constraints Archive, *11 data sets for the Frequency Assignment Problem*. With permission of "Centre d'Electronique de l'Armement", ftp://ftp.cs.city.ac.uk/pub/constraints/benchmarks/celar/

CPLEX OPTIMIZATION, Inc. (1995) *Version 4.0 of Using the CPLEX Callable Library and CPLEX Mixed Integer Library*, CPLEX Optimization, Inc., Suite 279, 930 Tahoe Blvd., Bldg. 802, Incline Village, NV 89451-9436, USA.

Crowder. H. P, Dembo. R. S and J. M. Mulvey, (1978) Reporting Mathematical Experiments in Mathematical Programming, *Mathematical Programming* 15: pp 316 - 329, North Holland Publishing Company.

Dincbas. M and Simonis. H, (1991), APACHE- A Constraint Based Automated Stand Allocation System, *Proceedings of the Advanced Software Technology in Air Transport Conference*, pp 267-282, London, England, October 1991.

Duncan. T, (1994) Schedule IT: Experiences with a Constraint Based Approach to Building an Intelligent Vehicle Scheduling System, presented at Constraint Handling Techniques Seminar, OR Society, London, June 1994.

El-Sakkout, (1995) Modelling Fleet Assignment in a Flexible Environment, *Proceedings of the Second International Conference on the Practical Application of Constraint Technology (PACT 96*), pp 27-39, The Practical Application Company Ltd, 1996.

European Computer Research Centre, Munich (1994), *ECLiPSe User Guide*, ECRC GmbH, Arabellastr. 17, Munich, Germany, pp 34 - 35.

Fisher. M and Jaikumar. R (1981), A Generalized Assignment Heuristic for the Large Scale Vehicle Routing Problem, *Networks*, 11:2, pp 109 - 124.

Hajian. M. T, El-Sakkout. H, Wallace. M, Lever. J. M, and Richards. E. B, (1995), Towards a Closer Integration of Finite Domain Propagation and Simplex-Based Algorithms, *Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics*, Florida, January 1996, also at http://www-icparc.doc.ic.ac.uk/papers

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

Little. J and Darby-Dowman. K (1995), The Significance of Constraint Logic Programming to Operational Research, *Operational Research Tutorial Papers*, 1995, Eds Lawrence. M and Wilsdon. C, pp 20 - 45.

Martello. S and Toth. P (1981), An Algorithm for the Generalized Assignment Problem, *Operational Research '81*, Ed Brans. J. P, pp 589-603, North-Holland, Amsterdam.

Martello. S and Toth. P (1990), *Knapsack Problems*, John Wiley and Sons Ltd, Chichester, England.

MIPLIB (1996) http://www.caam.rice.edu/~bixby/miplib/miplib.html

Numerical Algorithms Group Ltd and Brunel University, (1995), *FortMP Manual, Release 1*, NAG Ltd, Wilkinson House, Jordan Hill Road, Oxford, UK.

Ross. G. T and Soland. R. M. (1977), Modelling Facility Location Problems as Generalised Assignment Problems, *Management Science*, 24:3 pp 345 - 357.

Smith. B, Brailsford. S, Hubbard. P. M and Williams. H. P (1995), The Progressive Party Problem: Integer Linear Programming and Constraint Propagation Compared, *Principles and Practice of Constraint programming - CP'95, Proceedings of the First International Conference, CP'95 Cassis, France, September 19-22*, pp 36 - 52, Springer-Verlag Berlin Heidelberg.

Stella. F, Vercellis. C and Zaffalon. M (1994), A GAP Formulation of Production Planning Problems in Reconfigurable Assembly Lines, *Operations Research Proceedings 1994*, pp 334 - 338, Springer-Verlag, Berlin, Germany.

Shtub. A (1989), Modelling Group Technology Cell Formation as a Generalized Assignment Problem, *International Journal of Production Research*, 27: 5, pp 775 - 782.

Van Hentenryck. P and Carillon, J (1988), Generality versus Specificity: an Experience with AI and OR Techniques, American Association for Artificial Intelligence (AAAI-88), St Paul, Minnesota, August 1988.