LOGIC CUTS FOR MULTILEVEL GENERALIZED ASSIGNMENT PROBLEMS

María A. Osorio
School of Computer Science
Autonomous University of Puebla, Puebla 72560, México
aosorio@cs.buap.mx

Manuel Laguna
Leeds School of Business
University of Colorado, Boulder, CO 80309-0419, USA
laguna@colorado.edu

Abstract

In the multilevel generalized assignment problem (MGAP) agents can perform tasks at more than one efficiency level. Important manufacturing problems, such as lot sizing, can be easily formulated as MGAPs; however, the large number of variables in the related 0-1 integer program makes it hard to find optimal solutions to these problems, even when using powerful commercial optimization packages. The MGAP includes a set of knapsack constraints, one per agent, that can be useful for generating simple logical constraints or logic cuts. We exploit the fact that logic cuts can be generated in linear time and can be easily added to the model before solving it with classical branch and bound methodology. We generate all contiguous 1-cuts for every knapsack in large MGAP's problems and report the effect of adding these cuts in our experimental results.

Keywords: Branch and Bound, Generalized Assignment Problem, lot sizing, knapsack constraints, logic cuts.

1. Introduction

The multilevel generalized assignment problem (MGAP) was first described by Glover, Hultz, and Klingman (1979) in the context of large-scale task allocation in a major manufacturing firm. The same problem was addressed later by Laguna, *et al* (1995), who tackled the problem with a tabu search procedure that employed ejection chains to define neighborhoods of effective moves without significantly increasing the computational effort. To the best of our knowledge, the MGAP has not been addressed anywhere else in the literature.

A considerably large body of literature, however, exists for the classical Generalized Assignment Problem (GAP). A sample of exact methods is Ross and Soland (1975), Martello and Toth (1981), Fisher, Jaikumar and Van Wassenhove (1986), and Guignard and Rosenwein (1989). A survey by Cattrysse and Van Wassenhove (1992) provides a comprehensive examination of most of these methods.

The classical GAP consists of of assigning n tasks to m agents. Each task j must be assigned to one and only one agent i. Each agent i has a limited amount b_i of a single resource. An agent i may have more than one task assigned to it, but the sum of the resource requirements for these tasks must not exceed b_i. The resource requirements of a particular task depend on the agent to which the task is assigned, and they are denoted by a_{ij} (*i.e.*, the resources used by task j assigned to agent i). The cost of assigning task j to agent i is represented by c_{ij}

In addition to the exact procedures, researchers have developed heuristics for the GAP. Cattrysse, *et al.* (1994) developed a heuristic based on set partitioning, Amini (1995) reported results of a hybrid heuristic that combines two previous heuristics the GAP and the Variable-Depth-Search, Lorena and Narciso (1996) used a heuristic based on relaxing an IP formulation, and Chu and Beasley (1997) and Wilson (1997) utilized genetic algorithms. Even though these heuristics have been able to obtain high quality solutions to problems of medium and large size, their performance has been recently overshadow by the emergence of highly effective exact procedures. Savelsbergh (1995) developed a Branch and Price algorithm using a column generation approach and Nauss (1999) used linear programming cuts, lagrangean relaxation and subgradient optimization to reduce the solution time. Both of these procedures are capable of solving problems to optimality, which were previously considered insolvable.

The Multilevel Generalized Assignment Problem (MGAP), complicates the classical GAP problem, with the introduction of different levels of efficiency associated with each agent for performing each task. Now, the n tasks can be assigned to m agents with a maximum of l efficiency levels. Each task j must be assigned to one and only one agent i, but at a level k. Again, each agent i has a limited amount b_i of a single resource and the sum of the resource requirements for these tasks must not exceed b_i. The resource requirements of a particular task depend on the agent and level to which the task is assigned, and they are denoted by a_{ijk} (*i.e.*, the resources used by task j assigned to agent i at level k). Here, the cost of assigning task j to agent i at level k is represented by c_{ijk}. In real-world problems, the relationship between cost and resource utilization for any agent-level-task assignment is such that if $a_{ijk'} < a_{ijk''}$ then $c_{ijk'} > c_{ijk''}$. The objective of this combinatorial optimization problem is to minimize the total assignment cost. A 0-1 integer programming formulation of the MGAP follows:

$$\text{Minimize} \qquad Z(x) = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{l} c_{ijk} x_{ijk}$$

$$\text{subject to} \qquad \sum_{i=1}^{m} \sum_{k=1}^{l} x_{ijk} = 1 \qquad j = 1, \ldots, n \qquad (1)$$

$$\sum_{j=1}^{n} \sum_{k=1}^{l} a_{ijk} x_{ijk} \leq b_i \qquad i = 1, \ldots, m \qquad (2)$$

$$x_{ijk} \in \{0,1\}, \qquad i = 1, \ldots, m; \quad j = 1, \ldots, n; \quad k = 1, \ldots, l$$

In this model, the binary variable x_{ijk} is defined to be 1 if task j is assigned to agent i at the k^{th} level, and 0 otherwise. In the manufacturing application presented by Glover, Hultz, and Klingman (1979), the objective is to minimize the combined cost of production and inventory holding by determining an optimal product lot size and an optimal assignment of production to machines. There is a maximum of l possible lot sizes, and the machines work in parallel at different rates and operational costs. Some general-purpose machines are capable of producing several (or all) of the products while others are more specialized. For this application, c_{ijk} represents the combined setup, production, and holding cost (per unit time) incurred when product j is assigned to machine i using the k^{th} possible lot size. Therefore, for a particular product-machine pair, a small lot size results in a large combined cost and vice versa.

The MIP formulation of the MGAP consists of two sets of constraints. Set (1) consists of choice constraints, while set (2) consists of knapsack constraints. The knapsack constraints can be used to generate simple logical constraints that are similar to those in set (1). These simple constraints named logic cuts by Hooker *et al* (1994), and used extensively by Hooker and Osorio (1999) in several applications, have been shown to be effective in guiding B&B procedures to optimal solutions of hard optimization problems. They can be generated in linear time and when added to a model they shorten the time needed to solve problem instances using classical branch and bound methodology.

Specialized solution methods typically take advantage of the structure of the problem that they are trying to solve. One way of exploiting structure in integer programming is to identify strong cuts for a particular class of problems, such as cuts that define facets of the convex hull of integer solutions. An analogous approach can be used in logic-based methods (see Hooker, 1994). Logic-based methods often provide numerous opportunities to exploit structure (Bollapragada, 1995). While the identification of strong polyhedral cuts requires analysis of the convex hull (as in Hooker, 1992 a), and this can be a difficult task even for relatively simple problems, the identification of logic cuts need not involve polyhedral issues. One may be able to state useful logical relationships among integer variables in problems whose convex hull is far too complex to analyze (see Hooker and Osorio, 1999). In fact, some 0-1 facet-defining cuts are often discovered even in simple problems by first identifying logical relationships and then writing inequalities to capture the relationships, while others can be found analyzing the mathematical structure of the MIP model. Recently, the declarative use of logic cuts, derived from a knapsack constraint in a branch and cut framework to prune the searching tree, was succesfully applied by Osorio and Mújica (1999) to location problems.

This paper is organized as follows. Section 2 introduces the definition and properties of logic cuts, the characteristics that make knapsack constraints amenable to logic-cut generation and the procedures to generate valid cuts. Section 3 describes the computational experiments that have been conducted and finally, Section 4 presents the conclusions derived from this research.

2. Logic Cuts and Knapsack Constraints

According to the definition of Hooker *et al*. 1994, a logic cut is a constraint on the values of the integer variables that does not change the projection of the problem's epigraph onto the space of continuous variables. Furthermore, a logic cut has this property for any set of objective function coefficients, provided the integer variables have nonnegative coefficients. Logic cuts therefore cut off integer points that are dominated by others.

This definition is partially motivated by the work of Jeroslow (1984), who viewed integer variables as artificial variables used solely to define the shape of the epigraph in continuous space. From this perspective it is natural to admit cuts that leave the problem in continuous space undisturbed even if they cut off feasible solutions in the original space. A logic cut for a MILP model has therefore been characterized as an implication of the constraint set. Actually any logical formula implied by the constraint set as a whole is a logic cut, and a logic cut is true if it satisfies the constraints. A logic cut may be added to the problem without changing the optimal solution, but it may exclude feasible solutions (see Hooker *et al*., 1994).

A logic cut can also be seen as an extended clause. Extended clauses of degree k can be written as:

$$\sum_{j \in J} L_j \geq k,$$

where every L_j is a literal. Here, the sum is not arithmetic, because only true literals are taken into account. They are a useful compromise between arithmetic and logic because they express the notions of "at least" and "at most" and can be efficiently processed as logical formulas or expressed as binaries inequalities in a declarative form. It is important to note that Linear Programming is a stronger inference algorithm for extended clauses than unit resolution. For example, LP detects the infeasibility of the following inequalities, but unit resolution can do nothing with the corresponding extended clauses:

$$y_1 + y_2 + y_3 = 2$$
$$(1 - y_1) + (1 - y_2) + (1 - y_3) = 2$$

No known inference algorithm has exactly the same effect as LP on extended clauses, unless one views LP algorithms as inference algorithms.

An intuitive understanding of a problem can suggest logic cuts, both valid and nonvalid, even when no further polyhedral cuts are easily identified. The idea of a *(possibly nonvalid)* logic cut was defined by Hooker *et al.* (1994), who uses process synthesis as an example. Other examples include structural design problems in Bollapragada *et al* (1995), and a series of standard 0-1 problems discussed by Wilson (1990).

Whereas a cut in the traditional sense is an inequality, a logic cut can take the form of any restriction on the possible values of the integer variables, whether or not it is expressed as an inequality. Logic cuts can therefore be used to prune a search tree even when they are not expressed as inequality constraints in an MIP mode. But they can also be imposed as

inequalities within an MIP model, in which case they can tighten the linear relaxation and cut off fractional solutions as traditional cuts do.

The 0-1 knapsack constraints in set (2) of the problem can be expressed in the form $\boldsymbol{dy}=d$, where each $y_j \in \{0,1\}$. We can process these constraints logically using a complete inference algorithm for knapsack constraints developed by Hooker (1992-b). However this way of processing knapsack constraints presents difficulties, such as the conversion from an inequality constraint into its corresponding logical form. The most straightforward conversion is to write it as an equivalent set of logical propositions, but the number of propositions can grow exponentially with the number of variables in the inequality. Due to these difficulties, knapsack constraints are often used to generate logic cuts, in the form of extended inequalities, which can be easily manipulated. The logical clauses implied by a knapsack constraint are identical to the well-known "covering inequalities" for the constraint, and their derivation is straightforward (see Granot and Hammer, 1971).

While it is hard to derive all the extended inequalities implied by a 0-1 knapsack constraint, it is easy to derive all *contiguous cuts*. Consider a 0-1 inequality $\boldsymbol{dy} \geq d$ for which it is assumed, without loss of generality, that $d_1 \geq d_2 \geq \ldots \geq d_n > 0$. Note that if $d_j < 0$, its sign is reversed and d_j is added to δ. A contiguous cut for $\boldsymbol{dy} = d$ has the form,

$$\sum_{j=t}^{t+w+k-1} y_j \geq k, \tag{3}$$

where k is the degree of the cut and $w < n$ is the "weakness" (with $w = 0$ indicating a cut that fixes all of its variables). In particular (3) is a *t-cut* because the first term is y_t and it is valid if and only if

$$\sum_{j=1}^{t+k-1} d_j + \sum_{t+w+k}^{n} d_j < d, \tag{4}$$

Furthermore, Hooker and Osorio (1999) showed that every t-cut of weakness w for $dy \geq \delta$ is implied by a 1-cut of weakness w. Therefore, generate 1-cuts can be equivalent to generate all t-cuts in terms of infering values for the binary variables. In order for the paper to be self contained, we show in Fig.1 the algorithm developed by Hooker and Natraj (1999) that generates the 1-cuts we used for the MGAP problem. These cuts are generated in linear time. To illustrate, consider the knapsack constraint

$$13\,y_1 + 9\,y_2 + 8\,y_3 + 6\,y_4 + 5\,y_5 + 3\,y_6 \geq 30$$

This knapsack constraint gives rise to the following cuts,

$$y_1 + y_2 \geq 1$$
$$y_1 + y_2 + y_3 \geq 2$$
$$y_1 + y_2 + y_3 + y_4 + y_5 \geq 3.$$

Note that the first cut becomes redundant, once the second one is formulated.

Let $k = 1$, $s = \sum_{j=1}^{n} d_j$, $k_{last} = 0$.
For $j = 1, \ldots, n$:
  Let $s = s - d_j$.
  If $s < \delta$ **then**
    While $s + d_k < \delta$:
      Let $s = s + d_k$,
      Let $k = k + 1$.
    If $k > k_{last}$ **then**
      Generate the cut $y_1 + \ldots + y_j \geq k$.
      Let $k_{last} = k$.

Fig. 1 *An algorithm for generating all* 1*-cuts for a knapsack constraint* $dy \geq d$ *in which* $d_1 \geq d_2 \geq \ldots \geq d_n > 0$.

3. Experimental Results

Optimization algorithms for the classical GAP are generally tested on four classes of random problems, referred to as A,B,C and D (see Martello and Toth, 1981; Guignard and Rosenwein, 1989; Savelsbergh, 1997 for a detailed definition). Problems A, B and C are generated with independently uniform distributions for costs and resources, such that individual coefficients for costs and resources are not correlated. This leads to relatively easy problems where binary variables with small costs and small resource coefficients tend to be equal to 1, while binary variables with large costs and large resource coefficients tend to be equal to 0. These problems, even when large (Ross and Soland, 1975 solved problems with 4000 binary variables), can be solved in small number of nodes and consequently small times by commercial optimization packages using "generic" branch and bound. Problems type D (first introduced by Martello and Toth, 1981), inversely correlate individual costs and resources coefficients, yielding considerably more difficult problem instances.

Laguna, *et al.* (1995) developed for MGAP problems a more challenging and structurally different random problem generator, labeled E, that draws resource requirements from an exponential distribution with the costs coefficients inversely correlated. This probability function more accurately captures the disparity among equipment working at several levels in actual production facilities, where the setup time for highly specialized machines greatly differs from the time taken to prepare general purpose machinery.

The following is a description of the problem generator E used to create the random instances of MGAPs tested in this paper.

$a_{ijk} = 1.0 - 10\ ln(\text{Uniform}[0,1])$ with probability p, and with probability $1 - p$ if arc (i,j,k) is not included.

$c_{ijk} = 1000/a_{ijk} - 10\ \text{Uniform}[0,1]$

$b_i = \max\ (0.8 \sum_{j=1}^{n} \sum_{k=1}^{l} a_{ijk}) /m\ ,\ max_{\forall j,k}\ a_{ijk})$

The algorithm for generating the logic cuts was coded in C++ and uses CPLEX 6.5 for solving the resulting integer programming problem. We compare our results with the results obtained from solving the original integer programming formulation using the same CPLEX version. A Pentium Dell (100 MHz, with 32 Mb of RAM) was used for the computational work. We chose CPLEX results as a basis for comparison due to the lack of an exact algorithm for the MGAP, which to the best of our knowledge does not exist. Also because, CPLEX's latest release includes the generation of covering, clique and general upper bound cuts that are effective in solving "hard" optimization problems, strongly improving performance from previous versions.

It is interesting to point out that the relationship between the right-hand-sides and the sum of the coefficients for every knapsack in an E-generated instance is in the range where the contiguous cuts have greater impact due to the tightness of the resulting constraints (see Osorio and Mújica [1999]).

The first set of experiments focuses on assessing the performance of the proposed methodology on instances of the MGAP with several *p*-values that preserve the average number of variables in the problem. The objective was to study the impact of logic cuts in several combinations of number of tasks, agents, and levels. For these instances we reported the same E dataset used by Laguna, *et al.* (1995). The dataset includes 9 sets of 10 problems with numbers of agents and levels ranging from 3 to 5, numbers of tasks ranging form 20 to 40, and *p*-values of 1, 0.75, and 0.5.

SET	*p*-*level*	Number of			Number of		Number of Logic Cuts Added	Average %saved over CPLEX with Logic Cuts	
		Tasks	Agents	Levels	Variables	Constraints		CPU Secs	Nodes
1	1	20	3	5	300	23	59.2	-25.4	56.1
2	1	20	5	3	300	25	63.6	40.8	82.3
3	1	20	4	4	320	24	65.8	74	88.9
4	0.75	25	3	5	281	28	62.6	-13.1	36.5
5	0.75	25	5	3	281	30	61.8	43.9	87.8
6	0.75	25	4	4	300	29	65	20.7	72.9
7	0.5	40	3	5	300	43	79.4	-33.3	30.6
8	0.5	40	5	3	300	45	73.4	32.3	76
9	0.5	40	4	4	320	44	80.2	25.9	62.5

Table 1 *Summary results for Laguna, et al. (1995) instances*

The average CPU time for solving the instances with logic cuts was less than 1000 seconds and in the sets 1,4 and 7, where CPLEX performed faster without the cuts, the average CPU time was less than 100 seconds. This situation can be explained by the fact that the cuts can enlarge the problem size by a factor of 2 or 3, and even if their addition always reduces the number of nodes, a solution time advantage cannot always be obtained in problems that are solved in very few nodes. Note that problems with a larger ratio of agents/levels seem to obtain more reduction in CPU times with the addition of logic cuts, and that the *p*-level and the number of constraints do not seem to have much impact on the solution time.

In Table 2, we summarize the results of 9 sets of 10 larger E problems with the number of binary variables ranging from 720 to 3600 and a p value of 1. For these instances with full matrices, we chose various sizes of the number of tasks, agents and levels, with the objective of making a broader test of the merit of using logic cuts. We used 40 tasks for 8 sets and generated 60 task instances for the 9th set. The number of agents varied between 7 to 30 and the number of performance levels for each agent ranged between 2 and 4. The table includes the average of number of logic cuts added, CPU seconds, and number of nodes needed to prove optimality with the proposed methodology. To measure the merit of the logic cuts, in the column labeled "# Problems solved NO CUTS", we show the number of problems that CPLEX alone solved to optimality in less than 12 hours. The last two columns of Table 2 show the percentage savings in CPU seconds and nodes obtained with logic cuts.

	Number			Number of		Number of Logic Cuts	#Problems Solved	Averages with Logic Cuts		Average %saved over CPLEX with Logic Cuts	
SET	Tasks	Agents	Levels	Variables	Constraints	Added	NO CUTS	CPU Secs	Nodes	CPU Secs	Nodes
1	40	9	2	720	49	147	9	913.2	2291	39.16	89.66
2	40	10	2	800	50	162	6	3228.4	8619	72.02	86.18
3	40	12	2	960	52	194	5	6718.9	12586	38.79	87.86
4	40	15	2	1200	55	250	2	4003.9	6212	89.81	96.48
5	40	20	2	1600	60	1473	9	752.6	1145	43.73	83.94
6	40	8	3	960	48	177	8	1563.9	3257	10.65	83.37
7	40	10	3	1200	50	217	6	4008.2	5737	64.91	89.57
8	40	7	4	1120	47	188	9	558.3	1268	16.84	87.17
9	60	30	2	3600	90	649	0	8617.4	4445	NA	NA

Table 2 *Summary of results for large MGAP instances*

Note that while all problems were solved to optimality using logic cuts, CPLEX failed to solve 36 out of 90 instances within the 12 hours time limit. We calculate the average CPU time in seconds and the number of nodes for examples with logic cuts, using all the instances for each set, and the average percentage saved in CPU seconds and number of nodes, using only the results of problems solved to optimality with both models.

The CPLEX default parameters were used for both models with one exception. In the model with logic cuts we used an up branch first selection at each node. This strategy improved the solution time in the model with logic cuts, because of the special structure of the logic cuts. However it did not yield the same results in the model without logic cuts. We allowed a relative tolerance of 0.0001 on the gap between the best integer objective and the objective of the best node remaining. This tolerance is smaller than the tolerances used to solve the GAP problems reported in Savelsbergh (1995) and Nauss (1999).

It can be observed in Table 2 that the number of binary variables strongly impact the CPU seconds needed to solve problems to optimality. In addition to the number of variables, there are factors such as the ratio between agents and levels that strongly affect the solution time. Instances with smaller ratios of *agents/levels* yield more accurate logic

cuts while the logic cuts addition in problems with smaller ratios of *tasks/(agents*levels)* originate more clique variable elements. In fact the addition of logic cuts in problems with a ratio of *tasks/(agents*levels)* much smaller than one, generate problems that can be trivially solved. For ratio values close to one, typically, the number of clique members reaches 40% of the variables, making 'hard' problems easier to solve when the logic cuts are added.

In Table 3, we presented the instances for which CPLEX either could not prove optimality or could not reach the optimal solution in more than twelve hours of computer time. For this experiment, we used the time needed by our model to reach optimality, as a limit in the number of seconds that we allowed the problems to run with CPLEX alone. The main objective was to compare the best value that CPLEX alone could reach in the same CPU time needed by the model with logic cuts.

SET	Example Number	Logic Cuts Added	Optimal Value	CPLEX Best Found	Absolute Error	SET	Example Number	Logic Cuts Added	Optimal Value	CPLEX Best Found	Absolute Error
1	9	154	3320	3324	4	5	9	332	1469	1487	18
2	1	169	2850	2860	10	6	3	171	3757	3762	5
	2	161	3326	3328	2		8	174	3377	3377	0
	3	158	2963	2970	7	7	4	207	2851	2854	3
	5	163	2855	2866	11		5	222	2597	2605	8
3	1	185	2693	2705	12		9	218	2733	2736	3
	2	211	2297	2318	21		10	215	2735	2739	4
	4	193	2483	2491	8	8	4	186	3678	3688	10
	5	192	2407	2413	6	9	1	684	1731	1743	12
	6	197	2820	2846	26		2	649	1935	1946	11
4	1	245	1981	1990	9		3	661	1919	1919	0
	2	267	1720	1749	29		4	631	1983	1999	16
	3	251	1869	1869	0		5	617	2064	2079	15
	4	243	1933	1938	5		6	662	1819	1821	2
	5	248	2022	2033	11		7	674	1839	1844	5
	6	249	1980	1995	15		8	627	2092	2130	38
	8	261	1654	1658	4		9	644	2064	2092	28
	9	244	2022	2046	24		10	640	2018	2071	53

Table 3 *Best objective value found for models without logic cuts*

Tables 2 and 3 show that without logic cuts, CPLEX could solve only 54 of the 90 problems tested. Thanks to the logic cuts, the number of nodes was reduced by more than 80% in all the cases and the average CPU time saved in problems that CPLEX could solve in less than 12 hours, is in the range of 10.65% to 89.81%. Table 3 also shows the absolute difference between the solution found by CPLEX with and without the logic cuts. Note that out of the 54 problems for which the model without logic cuts could not prove optimality, only in 3 instances the optimal solution had been found within the time limit.

In our final experiment we used the model with logic cuts to obtain the optimal solution to a real-world lot sizing problem. The problem is to find the optimal lot sizes for 30 jobs that are to be processed on 7 machines with equal capacity of 2106 units. Each job may be processed in up to 3 different lot sizes. This problem results in an integer programming formulation with 338 binary variables (*i.e.*, the coefficient matrix is approximately 54% full) and 37 constraints (see Laguna, *et al.*, 1995 for the complete problem data).

The best known solution for this problem, found by Laguna, *et al.* (1995) using tabu search has an objective function value of 691,634. The Logic Cuts added to the problem allowed CPLEX to solve the problem in a CPU time of 36 hours, with 1,990,161 nodes, and to find a solution with an objective function value of 690,624 with a relative optimality tolerance of 0.01. We used CPLEX, with the same tolerance, to solve the model without the cuts, but after 18 hours of CPU time, the B&B tree with 2,180,238 was consuming 357 Mb and the execution was aborted because the computer was out of memory. The best solution before aborting was 691,478, which is much worse than our solution. It is interesting to note that even if the tabu search code found the 691,634 solution in only 120.07 CPU seconds, it was shown that the heuristic could not improve this suboptimal value even if allowed to run the same 36 hours.

4. Conclusions

In the course of this research, we have identified that the IP model of the MGAP is a suitable framework for the generation of logic cuts from knapsack constraints. The desegregation of the variables in the problem leads to the generation of accurate cuts with fewer variables. The contiguous cuts used in the MILP model resulted in a large reduction in the number of nodes. For small problems, however, the reduction in the number of nodes does not always results in a reduction of the solution time. Therefore, adding logic cuts to small problems cannot be recommended from the point of view of reducing solution time. For large problems, the node reduction pays off, because the time grows exponentially with respect to the number of nodes in searching trees. Our results confirm that adding contiguous logic cuts to an IP model is an effective way of solving large instances of the MGAP. We have also experimented with the application of the contiguous logic cuts to the classical GAP and we concluded that there is no merit in adding such cuts.

A byproduct of our research was the ability to find a 99% optimal solution to a real world lot sizing problem. The solution found improves upon the previous best known solution and is also better than the best found solving the model without the logic cuts.

A promising direction for future research consists of finding effective strategies for selecting a subset of the logic cuts instead of adding the entire set. This will yield smaller models and in turn further reduce solution time.

References

Amini, M and M. Racer (1995). "A hybrid heuristic for the generalized assignment problem", *European Journal of Operational Research*, **5:2**, 343-350.

Bollapragada, S., O. Ghattas and J. N. Hooker (1995). "Optimal Design of Truss Structures by Mixed Logical and Linear programming", *manuscript*, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh.

Cattrysse, D., and L. Van Wassenhove (1990). "A Survey of Algorithms for the Generalized Assignhment Problem", *European Journal of Operational Research*, **46**, 84-92.

Cattrysse, D., M. Salomon, and L. Van Wassenhove (1994). "A Set Partitioning Heuristic for the Generalized Assignment Problem", *European Journal of Operational Research*, **72**, 167-174.

Chu, P.C. and J.B. Beasley (1997). "A Genetic Algorithm for the Generalized Assignment Problem", *Computers and OR*, **24**, 17-23.

Fisher, M.L., R. Jaikumar, and L.N. Van Wassenhove (1986). "A Multiplier Adjustment Method for the Generalized Assignment Problem", *Management Science* **32:9**, 1095-1103

Glover, F., H. Jultz, and D. Klingman (1979). "Improved Computer-Based Planning Techniques – Part II", *Interfaces*, **9:4**, 12-20.

Granot, F., and P. L. Hammer (1971). "On the use of boolean functions in 0-1 linear programming", *Methods of Operations Research*, 154-184.

Guignard, M. and M. Rosenwien (1989). "An Improved Dual Based Algorithm for the Generalized Assignment Problem", *Operations Research*, **37:4**, 658-663.

Hooker, J. N. (1992 a). "Logical inference and polyhedral projection", Proceedings, Computer Science Logic Workshop (CSL'91), *Lecture Notes in Computer Science* **626**, 184-200.

Hooker, J. N. (1992 b). "Generalized resolution for 0-1 linear inequalities", *Annals of Mathematics and AI* **6**, 271-286.

Hooker, J.N. (1994). "Logic-based methods for optimization", in A. Borning, ed., Principles and Practice of Constraint Programming, *Lecture Notes in Computer Science* **874**, 336-349.

Hooker, J.N., H. Yan, I. Grossmann, and R. Raman (1994). "Logic cuts for processing networks with fixed charges", *Computers and Operations Research* **21**, 265-279.

Hooker, J.N., and N.R. Natraj (1999). "Solving 0-1 optimization problems with k-tree relaxation", *in preparation*.

Hooker, J.N., and M.A. Osorio (1999). "Mixed Logical/Linear Programming". *Discrete Applied Mathematics* **96-97**, 395-442

Jeroslow, R. E., and J. K. Lowe (1984). "Modeling with integer variables", *Mathematical Programming Studies* **22**, 167-184.

Jeroslow, R. E. (1987). "Representability in mixed integer programming, I: Characterization results", *Discrete Applied Mathematics* **17**, 223-243.

Laguna, M., J.P. Kelly, J.L. González-Velarde, and F. Glover (1995). "Tabu Search for the Multilevel Generalized Assignment Problem", *European Journal of Operational Research*, **82**, 176-189.

Lorena, L.A., N. Narciso, and G. Marcelo (1996)."Relaxation heuristics for a Generalized Assignment Problem". *European Journal of Operational Research*. **91:3**, 600-607

Martello, S., and P. Toth (1981). "An Algorithm for the Generalized Assignment Problem", *Proceedings of the 9th IFORS Conference*, Hamburg, Germany.

Nauss, R.M. (1999). "Solving the Classical Generalized Assignment Problem", *working paper*, School of Business Administration. U. of Missouri-St. Louis.

Osorio, M. A., and Rosalba Mújica (1999). "Logic Cuts Generation in a Branch and Cut Framework for Location Problems". To appear in *Investigación Operativa.*

Ross,G.T., and R.M. Soland (1975). "A Branch and Bound Algorithm for the Generalized Assignment Problem", Mathematical Programming, **8**, 91-103.

Savelsbergh, M. (1997). "A Branch-and-Price Algorithm for the Generalized Assignment Problem", *Operations Research*, **45:6**, 831-854.

Williams, H.P., and K.I.M. McKinnon (1989). "Constructing Integer Programming Models by the Predicate Calculus". *Annals of Operations Research*, **21**, 227-246.

Williams, H.P. (1995). "Logic applied to integer programming and integer programming applied to logic", *European Journal of Operational Research* **81**, 605-616.

Wilson, J. M. (1990). "Generating cuts in integer programming with families of specially ordered sets", *European Journal of Operational Research,* **46**, 101-108.