Logic Cuts Generation in a Branch and Cut Framework for Location Problems

María A. Osorio Lama *Rosalba Mújica García*

School of Computer Science
Autonomous University of Puebla, Puebla 72560
México

Abstract
In the warehouse location problem, the objective is to choose a set of warehouses of limited capacity so as to serve a set of demand points while minimizing fixed and transport costs. In particular, it includes a knapsack constraint that can be useful for generating simple logical constraints named logic cuts. The logic cuts can be generated in linear time and can be implemented in a branch and cut framework for accelerating the solution of the capacitated warehouse location problems with fixed costs. We use the Hooker algorithm for generating all contiguous t-cuts in every node of the tree and report the experimental results.

Keywords: Warehouse Location Problems, Knapsack Constraint, Logic Cuts.

1 Introduction

Capacitated warehouse location models arise in many practical applications. The MILP formulation includes a model that use binary variables to denote the existence of some of the total sources that accomplish the total demand and minimize the total cost. In this case, the cost includes the fixed existence cost for every warehouse and the continuous transportation cost for every combination between the warehouse sources and the demand points.

Unfortunately, the solution of these models can be quite expensive with the LP-based branch and bound methods that are implemented in many commercial and academic computer codes (e.g. OSL, MPSX, SCICONIC, ZOOM, LINDO, APEX, CPLEX, etc.) This has motivated in recent years substantial development of reformulation techniques and cutting plane algorithms, as for instance in the work by Sherali and Adams and Balas et al. [2]. However, these new developments have concentrated on numerical aspects, mainly on convexification with branch and bound.

Although these numerically based techniques are quite promising, an alternate and complementary direction is to exploit the logical structure of an MILP problem (see Jeroslow [23]). A problem may have logical constraints that restrict the number of solutions that need to be enumerated, although these constraints may not be explicit in the MILP model. Constrains of this kind, once thery are identified, can be used either as additional inequality constraints within the MILP model or as symbolic constraints that restrict the generation of alternatives in a branch-and-bound search.

In particular, the capacitated warehouse location model includes knapsack constraint that can be very useful for generating simple logical constraint with certain and very interesting properties. This logical constraints are named logical cuts and can be implemented in a branch and cut framework and can be used to accelerate the solution of the capacitated warehouse location problems with fixed costs.

2 The Capacitated Warehouse Location Problem

In the classical problem, the idea is to choose a set of warehouses of limited capacity so as to serve a set of demand points while minimizing fixed and transport costs. Let

x_{ij} = Flow from warehouse i to demand point j.

f_i = Warehouses Fixed Cost i.

k_i = Warehouse Capacity i.

d_j = Demand point j.

c_{ij} = Transportation Unit Cost from i to j.

The MILP model is

$\min \sum_i k_i y_i + \sum_{ij} c_{ij} x_{ij}$

s.a.

$\sum_j x_{ij} \leq k_i y_i \quad \forall i$

$\sum_i x_{ij} \geq d_j \quad \forall j$

$x_{ij} \geq 0 \quad \forall i, j$

$y_i \in 0, 1 \quad \forall i$

The capacitated warehouse location problem is very useful to illustrate how cuts can be generated from knapsack constraints.

3 Extended Clauses

The knapsack constraint is a extended clause. Extended clauses seem a particularly useful compromise between arithmetic and logic because they express the notions of "at least" and "at most" but can be efficiently processed as logical formulas. In fact, Barth's constraint-based solver for 0-1 optimization problems [3] reasons with 0-1 inequalities only after converting them to extended clauses.

An extended clause of degree k can be written

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

where each L_j is a literal. Here the sum is not an arithmetical sum but simply counts the number of literals that are true. Ordinary clauses have degree 1. To say that exactly k are true, one can write

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

and one can use two extended clauses to say exactly k are true.

A complete inference algorithm ("generalized resolution") for extended clauses was presented in [11,14] and is refined by Barth in [2]. It uses resolution as well as a diagonal summation, where the latter is defined as follows: An extended clause $\sum_{j\in J} L_j \geq k+1$ is the diagonal sum of the set of extended inequalities $\{\sum_{j\in J} L_j \geq k | i \in J\}$ if $J = \bigcup_{j\in J} J_i$ but for each $i \in J, i \notin J_i$.

The algorithm of [11] is applied to a set S of extended clauses as follows. If there are two clauses C_1, C_2 of degree 1 with a resolvent C that is implied by no extended clause in S, such that C_1 is implied by an extended clause in S and similarly for C_2, then add C to S. If there is a set E of extended clauses with a diagonal sum D that is implied by no extended clause in S, such that each clause in E is implied by some clause in S, then add D to S. The algorithm continues until no more clauses can be added to S. This can be done thanks to the following theorem:

Theorem 1 (Hooker [11],[14]). A set S of extended clauses implies clause C if and only if the generalized resolution algorithm applied to S generates a clause that implies C.

Implementation of the algorithm requires recognition of when one extended clause implies another. We say that $\sum_{j\in J1} L_{1j} \geq k_1$ implies $\sum_{j\in J2} L_{2j} \geq k_2$ if and only if

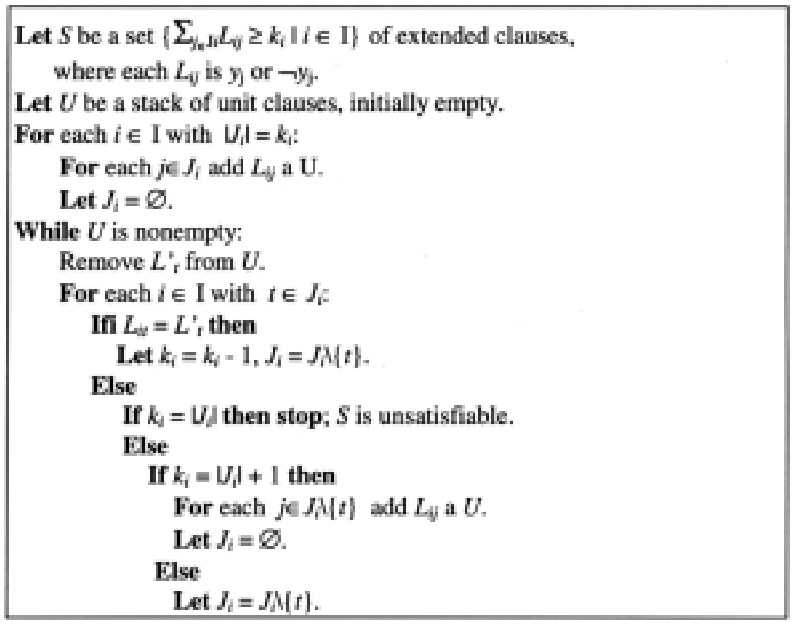
$$|J_1| - |\{j \in J_1 \cap J_2 | L_{1j} = L_{2j}\}| \leq k_1 - k_2.$$

When all of the literals of a derived extended clause are positive and correspond to sets of inequalities, a linear relaxation can be formulated. A unit resolution algorithm for extended clauses appears in Fig. 1.

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:

$$y1 + +y2 + y3 \geq 2$$
$$(1 - y1) + (1 - y2) + (1 - y3) \geq 2$$

No known inference algorithm has exactly the same effect as LP on extended clauses, unless one views LP algorithms as inference algorithms. Generalized resolution is of course stronger than LP.



```
Let $S$ be a set $\{\sum_{j \in J_i} L_{ij} \geq k_i \mid i \in I\}$ of extended clauses,
   where each $L_{ij}$ is $y_j$ or $\neg y_j$.
Let $U$ be a stack of unit clauses, initially empty.
For each $i \in I$ with $|J_i| = k_i$:
    For each $j \in J_i$ add $L_{ij}$ a U.
    Let $J_i = \varnothing$.
While $U$ is nonempty:
    Remove $L'_t$ from $U$.
    For each $i \in I$ with $t \in J_i$:
        Ifi $L_{it} = L'_t$ then
            Let $k_i = k_i - 1$, $J_i = J_i \setminus \{t\}$.
        Else
            If $k_i = |J_i|$ then stop; $S$ is unsatisfiable.
            Else
                If $k_i = |J_i| + 1$ then
                    For each $j \in J_i \setminus \{t\}$ add $L_{ij}$ a $U$.
                    Let $J_i = \varnothing$.
                Else
                    Let $J_i = J_i \setminus \{t\}$.
```

Fig. 1: A unit resolution algorithm for extended clauses.

4 Knapsack Constraints

A complete inference algorithm for knapsack constraints appears in [14], and an analog of unit resolution can easily be devised for them. Nevertheless, the knapsack constraints can be better used as a source of logic cuts that are easily processed, such as clauses and extended clauses. The implied clauses, for example, are identical to the well-known "covering inequalities" for the constraint, and their derivation is straighforward (e.g. [8]).

It may be more effective, however, to infer extended inequalities. Although it is hard to derive all the extended inequalities that are implied by a constraint, it is easy to derive all contiguous cuts. Consider a 0-1 inequality $d_y \geq \delta$ for which it is assumed, without loss of generality, that $d_1 \geq d_2 \geq ... \geq d_n > 0$; if $d_j < 0$, reverse its sign and add d_j to δ. A contiguous cut for $d_x \geq \delta$ is one of the form,

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

where k is the degree of the cut and w < n the "weakness" (w = 0 indicates a cut that fixes all of its variables). In particular (1) is a t-cut because the first term is y_t. (1) is valid if and only if

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

Furthermore,

Theorem 2 (Hooker [18]) Every t-cut of weakness w for $d_x \geq \delta$ is implied by a 1-cut of weakness w.

The power of all t-cuts can therefore be obtained by generating only 1-cuts. The algorithm of Fig. 2, presented in by Hooker in [18] does this in linear time. By way of example, the knapsack constraint,

$$13y1 + 9y2 + 8y3 + 6y4 + 5y5 + 3y6 \geq 30$$

gives rise to the 1-cuts,

$y1 + y2 \geq 1$

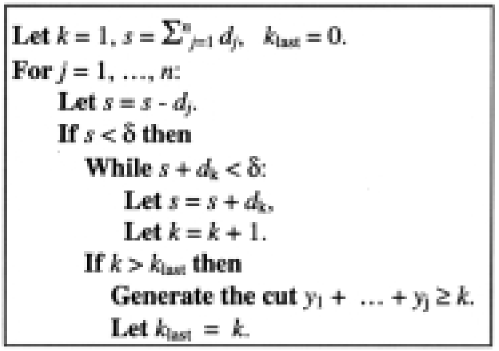
$y1 + y2 + y3 \geq 2$

$y1 + y2 + y3 + y4 + y5 \geq 3.$

The first cut could be deleted if desired, because it is redundant of the second.

5 Logic Cuts

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 in [21], which gives the process synthesis



```
Let $k = 1$, $s = \sum^{n}_{j=1} d_j$,  $k_{\text{last}} = 0$.
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 $k > k_{\text{last}}$ then
            Generate the cut $y_1 + \ldots + y_j \geq k$.
            Let $k_{\text{last}} = k$.
```

Fig. 2: An algorithm for generating all t-cuts for a knapsack constraint $d_y \geq \delta$ in which $d_1 \geq d_2 \geq ... \geq d_n > 0$;.

example as an example. Other examples include structural design problems [10], matching problems [32], and a series of standard 0-1 problems discussed by Wilson [73].

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 MILP mode. But they can also be imposed as inequalities within an MILP model, in which case they can tighten the linear relaxation and cut off fractional solutions as traditional cuts do.

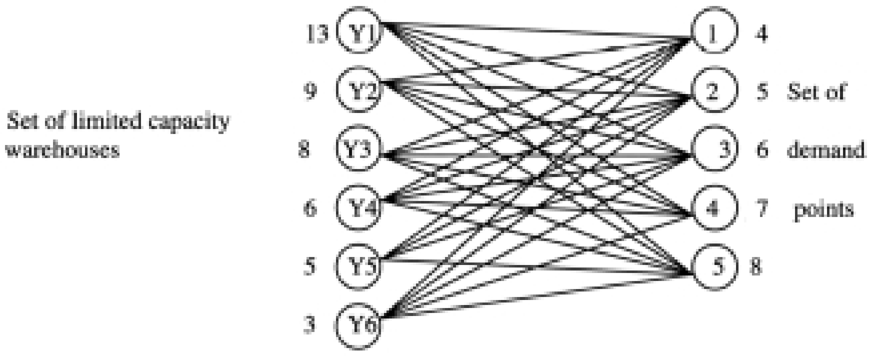
Taking the Hooker [21] definition of a logic cut, it will be a constraint on the values of the integer variables that does not change the projection of the problem's epigraph onto the space of continuous variables. Furthermore, a logic cut must have this property for any set of objective function coefficients, 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 [23], 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 undistrubed even if they cut off feasible solutions int he 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 satisfies the constraints even if is not implied by the constraints[19]. Logic cuts can be defined in an even more general sense that permits them to be nonvalid. A cut may be added to the problem without changing the optimal solution, but it may exclude feasible solutions.

6 Example

We have 6 warehouses of limited capacity and 5 demand points, the warehouses capacity is 13, 9, 8, 6, 5 y 3, and we have the following demand in every demand point: 4, 5, 6, 7, y 8, according to the following Figure:



The MILP model to represent this problem is:

Minimize

$250Y1 + 180Y2 + 170Y3 + 160Y4 + 140Y5 + 120Y6 +$
$50x11 + 55x12 + 60x13 + 65x14 + 70x15 +$
$55x21 + 50x22 + 55x23 + 60x24 + 65x25 +$
$60x31 + 55x32 + 50x33 + 55x34 + 60x35 +$
$65x41 + 60x42 + 55x43 + 50x44 + 55x45 +$
$70x51 + 65x52 + 60x53 + 55x54 + 50x55 +$
$75x61 + 70x62 + 65x63 + 60x64 + 55x65+$

Subject to:

$x11 + x12 + x13 + x14 + x15 \leq 13Y1$
$x21 + x22 + x23 + x24 + x25 \leq 9Y2$
$x31 + x32 + x33 + x34 + x35 \leq 8Y3$
$x41 + x42 + x43 + x44 + x45 \leq 6Y4$
$x51 + x52 + x53 + x54 + x55 \leq 5Y5$
$x61 + x62 + x63 + x64 + x65 \leq 3Y6$

$x11 + x12 + x31 + x41 + x51 \geq 4$

$x12 + x22 + x32 + x42 + x52 \geq 5$
$x13 + x23 + x33 + x43 + x53 \geq 6$
$x14 + x24 + x34 + x44 + x54 \geq 7$
$x15 + x25 + x35 + x45 + x55 \geq 8$

$13Y1 + 9Y2 + 8Y3 + 6Y4 + 5Y5 + 3Y6 \geq 30 \quad (Knapsack \quad Constraint)$

$x_{ij} \geq 0 \quad Y_i = 0, 1$

Using the classical Branch and Bound method, we obtain the following tree:

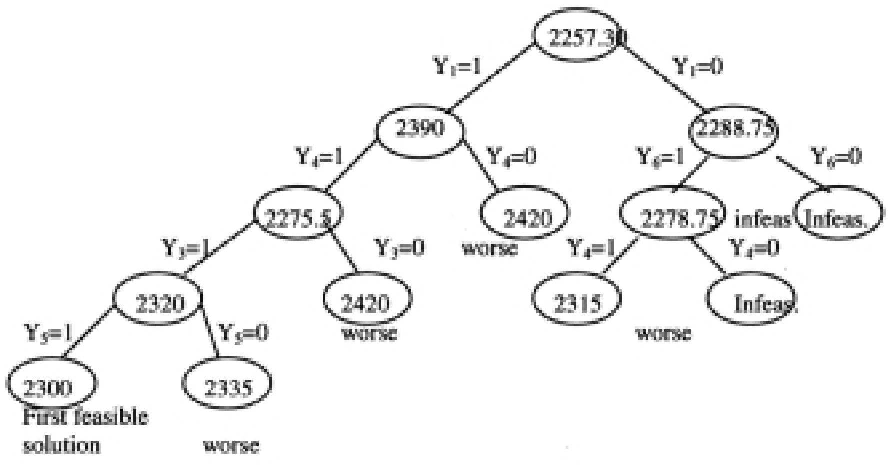


Fig. 3: Branch and Bound Tree for the example.

As we can see, we need 13 nodes to obtain the optimal solution. If we use the logic cuts generation in every node, we obtain the following branch and bound tree:

We can see that we only need 7 nodes to obtain the optimal solution. Notice that the knapsack constraint will be different in every node and that we need to generate the logic cuts from the current knapsack constraint at that node.

7 Experimental Results and Conclusions

The following table shows the CPU seconds and the nodes number for the classical branch and bound method and for the branch and logic cut framework. We show the results for the examples tested. The numerical tests were performed on a SUN Sparc Station with C programs that uses CPLEX Optimization Libraries.

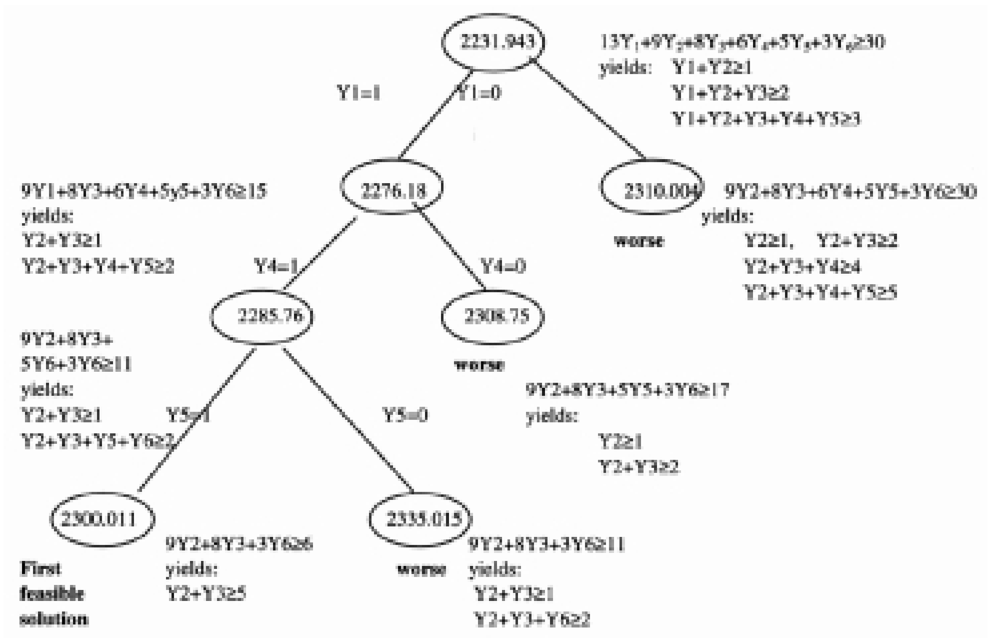


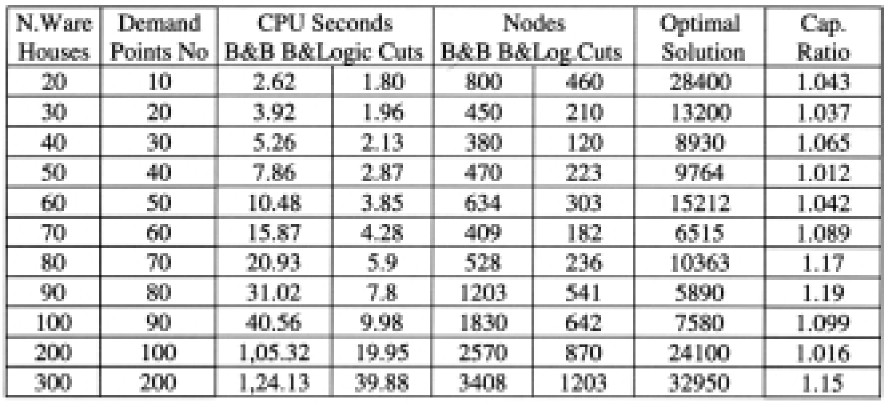
Fig. 4: Branch and Logic Cut Tree for the example.

The fixed costs were generated using a normal distribution with a mean of 200 and a standard deviation of 20; the variable costs, according to the function: $50+5*|i-j|$, where i and j are the warehouse and demand point indexes, respectively. The right hand sides for the capacity warehouse constraints were obtained with a normal distribution, using the number of warehouses as a mean and the half of this value as a standard deviation. For the demand amounts, we used numbers sequentially generated in order to accommodate the ratio tested of total warehouse capacity to total demand.

The warehouse location problem is a very good example for the generation of logic cuts from a knapsack constraint. The contiguous cuts used in the MILP model result in a 40%-60% reduction in the number of nodes and almost the same reduction in time. The results tend to confirm that the best way to solve the warehouse location problems is by adding contiguous cuts to the traditional MILP model.

Another important result, shown in the problems solved, is that contiguous cuts have greater have greater effect when the problem is more tightly constrained, as roughly indicated by the ratio of total warehouse capacity to total demand.

Because the logic cuts represent deeper cuts than the knapsack constraint in Mixed Integer Linear problems, to insert logic cuts in every node of the searching



N.Ware Houses	Demand Points No	CPU Seconds B&B	B&Logic Cuts	Nodes B&B	B&Log.Cuts	Optimal Solution	Cap. Ratio
20	10	2.62	1.80	800	460	28400	1.043
30	20	3.92	1.96	450	210	13200	1.037
40	30	5.26	2.13	380	120	8930	1.065
50	40	7.86	2.87	470	223	9764	1.012
60	50	10.48	3.85	634	303	15212	1.042
70	60	15.87	4.28	409	182	6515	1.089
80	70	20.93	5.9	528	236	10363	1.17
90	80	31.02	7.8	1203	541	5890	1.19
100	90	40.56	9.98	1830	642	7580	1.099
200	100	1,05.32	19.95	2570	870	24100	1.016
300	200	1,24.13	39.88	3408	1203	32950	1.15

tree allows us to obtain a basic feasible solution earlier in the tree. It is important to remark the fact that we need to obtain a new release, according to the integer variables fixed at that node of the tree, for the knapsack constraint, before generating the logic cuts for that node. Those cuts will remain in the model only for that specific node and its descendants, and we must remove them for backtracking the tree.

Because the time in a branch and bound tree is mainly spent in the solution of every linear model at each node of the tree, to decrease considerably the number of nodes, will have the same effect in the problem resolution time, as we can see in the problems solved. In a smaller searching tree the time spent in finding the optimal solution will be, of course, smaller. On the other hand, the time spent in generating logic cuts is very low because Hooker's algorithm can do it in linear time. Even if the addition of those new constraints to each node enlarges the linear problem to be solved, the increment is very low and we have the advantage that the logic cuts generated will remain in the model only for this specific node and its descendant.

References

[1] Balas, E., Disjunctive programming: Cutting planes from logical conditions, in O.L. Mangasarian, R. R. Meyer and S. M. Robinson, eds., Nonlinear Programming 2, Academic Press (New York, 1975) 279-312.

[2] Balas, E., Disjunctive programming, Annals Discrete Mathematics 5 (1979) 3-51.

[3] Barth, P., Logic-Based 0-1 Constraint Programming, Kluwer Academic Publishers (Boston, 1995).

[4] Beaumont, N., An algorithm for disjunctive programs, European Journal of

Operational Research 48 (1990) 362-371.

[5] Beasley, J. E., An algorithm for solving large capacitated warehouse location problems, European Journal of Operational Research 3 (1988) 314-325.

[6] Bollapragada, S., O. Ghattas and J. N. Hooker, Optimal Design of truss structures by mixed logical and linear programming, manuscript, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213 USA (1995).

[7] Chandru, V., and J. N. Hooker, Extended Horn sets in propositional logic, Journal of the ACM 38 205-221.

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

[9] Hammer, P.L., and S. Rudeanu, Boolean Methods in Operations Research and Related Areas, Springer Verlag (Berlin, New York, 1968).

[10] Hooker, J. N., Resolution vs. Cutting plane solution of inference problems: Some computational experience, Operations Research Letters 7 (1988) 1-7.

[11] Hooker, J. N., Generalized resolution and cutting planes, Annals of Operations Research 12 (1988) 217-239.

[12] Hooker, J. N., A quantitative approach to logical inference, Decision Support Systems 4 (1988) 45-69.

[13] Hooker, J. N., Input proofs and rank one cutting planes, ORSA Journal on Computing 1 (1989) 137-145.

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

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

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

[17] Hooker, J. N., Testing heuristics: We have it all wrong, Journal of Heuristics 1 (1995) 33-42.

[18] Hooker, J.N., and N.R. Natraj, Solving 0-1 optimization problems with k-

tree relaxation, in preparation.

[19] Hooker, J.N., and M.A. Osorio, Mixed Logical/Linear Programming. EDRC Report No. 02-08-1996. Carnegie Mellon University. To appear in Discrete Applied Mathematics.

[20] Hooker, J.N., and G. Rago, Partial instantiation methods for logic programming, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213 USA (1995).

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

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

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

[24] Raman, R., and I. E. Grossmann, Modeling and computational techniques for logic based integer programming, Computer and Chemical Engineering 18 (1994) 563-578.

[25] Williams, H.P., Logical problems and integer programming, Bulletin of the Institute of Mathematics and its Implications 13 (1977) 18-20.

[26] Williams, H.P., Linear and integer programming applied to the propositional calculus, International Journal of Systems Research and Information Science 2 (1987) 81-100.

[27] Williams, H.P., An alternative explanation of disjunctive formulations, European Journal of Operational Research 72 (1994) 200-203.

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

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