Solving Covering Problems Using LPR-Based Lower Bounds

Stan Liao (stanliao@synopsys.com)
Advanced Technology Group, Synopsys, Inc.

Srinivas Devadas (devadas@mit.edu)
Department of EECS, MIT

Abstract—Unate and binate covering problems are a special class of general integer linear programming problems with which several problems in logic synthesis, such as two-level logic minimization and technology mapping, are formulated. Previous branch-and-bound methods for exactly solving these problems use lower-bounding techniques based on finding maximal independent sets. In this paper we examine lower-bounding techniques based on linear programming relaxation (LPR) for the binate covering problem. We show that a combination of traditional reductions (essentiality and dominance) and incremental computation of LPR-based lower bounds can exactly solve difficult covering problems orders of magnitude faster than traditional methods.

Keywords—Covering problems, integer linear programming

I. Introduction

Covering problems (unate and binate) are important combinatorial optimization problems with which several problems in logic synthesis (such as two-level logic minimization [12], state minimization [6], exact encoding), and in code generation and optimization (e.g., [7]), are formulated. These problems are obviously NP-hard, and given their wide applicability it is of great practical interest to obtain approximate solutions of good quality in a tolerable amount of time. However, to evaluate the quality of heuristic solutions we will need to either obtain tight lower bounds or solve the problems exactly. Since covering problems are a special class of general integer linear programming (ILP) problems, previous research has focused on solving these problems using specialized techniques such as essentiality, row dominance, column dominance, Gimpel's reduction, and component reduction. Yet it is precisely for this reason that techniques for solving ILP have been largely ignored. In particular, the basic lower-bounding procedure for ILP, namely the linear-programming relaxation (LPR), has not been strategically applied to covering problems.

In this paper we show that LPR-based techniques yield lower bounds of quality superior to those obtained by previous methods, thereby allowing for termination of unsuccessful branches at earlier stages of the search tree. Although computing the optimal solution of linear programs corresponding to the covering problems is more costly than independent-set based techniques, experimental results demonstrate that the improved quality of lower bounds in fact makes it possible to prove optimality more quickly in some cases or obtains better solutions (when given a specific amount of time) in others. Further, incremental computation of LPR-based bounds can result in faster search. We also show that simply formulating a covering problem as a general integer linear programming problem sometimes cause difficulties for the ILP solver which may not discover covering-problem-specific reductions (i.e., essentiality, dominance, etc.).

In Section II we review previous methods for solving these problems. We then introduce our LPR-based lower bound computation in Section III, and present our solver BCU. In Section IV we present our experimental results and compare them with results obtained by previous methods as well as with results achieved by a commercial ILP solver.

II. Covering Problems

We refer the reader to [3] for the statement of the unate-covering (also called *set-covering*) and binate-covering problems. We briefly review previous work and summarize the contribution of this paper.

A. Previous Work

In VLSI-CAD research, approaches to solving covering problems have largely followed the pioneering work of Quine and McCluskey on logic minimization [8] [10] [11]. The basic idea is to reduce the size of the problem via essentiality, column dominance, and row dominance. Gimpel devised another form of reduction which is applicable when two columns have the same cost and the rows they cover show certain properties [5]. Reductions are applied successively until no more are possible, at which point a *cyclic core* is said to have resulted. The problem is then solved exactly with a branch-and-bound procedure, or heuristically with a no-backtrack, limited-backtrack, or limited-time version thereof. At each step, a column is selected (based on some heuristic measure) that is likely to minimize the cost and maximize the number of rows covered, and reductions are again applied. Additional techniques such as partitioning the problem into independent subproblems [12] can be readily incorporated. Recently, Coudert and Madre have discovered a new pruning condition (reduction), which is based on the *limit lower-bound* [4]. Refinement of their techniques by caching the solutions of subproblems led to further improvements [3]. It is noteworthy that, in this line of works, the lower-bounding procedure in the branch-and-bound is primarily based on *maximal independent set*. Although he has presented a theoretically better lower bound for unate problems, Coudert remarks that this lower bound does not always yield desired results. In addition, the effective limit lower-bound pruning make use of the independent set only, not the theoretically better one.

The set-covering problem has also been of interest to the operations research community. Beasley has presented a number of heuristics as well as lower-bounding techniques [2]. His lower-bounding techniques are based on solving the dual problem of the linear-programming relaxation (LPR), using a *dual ascent* procedure consisting of refining the dual solution while keeping it feasible, which, by the weak duality theorem, always provides a lower bound for the primal problem. However, his examples are apparently randomly generated with nonuniform cost functions, and his heuristics (dual ascent, Lagrangian, and subgradient) are aimed at problems with nonuniform costs. In practice, many problems in CAD have uniform or near-uniform cost functions. As we will show in Section III, in the case of uniform cost, the solution obtained by a dual ascent procedure is just an independent set. In addition, the binate-covering problem is not considered and reductions are not systematically applied.

Barth [1] solves a slightly more general problem than covering using an implicit enumeration approach. In his work he showed that his method equals or outperforms a commercial ILP solver for certain classes of problems. However, upon closer examination, the complicated mathematical treatment he has presented is simply an extension of the well-known reductions based on essentiality and dominance, and would provide no additional benefit for the covering problems we consider. In fact, we experimented with his program, and for covering problems it is actually much slower than Coudert's.

B. Our Contribution

The contribution of this paper is as follows. We first show that lower bounds obtained by linear-programming relaxation are in theory equal or superior to those obtained with maximal independent set (MIS). (Experimental results confirm, indeed, that the LPR-based lower bounds are almost always better.) However, using LPR bounds in a naïve fashion such as the method employed in commercial ILP solvers does not result in uniformly better performance than MIS-based solvers. We have developed efficient ways of incorporating LPR bounds into a binate covering framework, and we show that for many difficult examples we are able to obtain exact solutions 10–100× faster, or a solution of lower cost if we could not complete in the time limit given.

III. LPR-Based Lower Bounds

A. Covering Problem as ILP

A covering problem may be readily translated into an ILP as follows. We treat each variable x_j as an integer variable in the ILP, and

r_1	1		1		
r_2	1	1			
r_3		1	1	1	
r_4			1	1	1

Fig. 1. Rows r_2 and r_4 form an independent set since they do not intersect. In fact, they are the maximal independent set of this covering matrix.

substitute $(1 - x_j)$ for $\overline{x_j}$ in each clause. Then, we write a clause y_i as $y_i \geq 1$, and move the constants from the left hand side to the right. For instance, the clause

$$x_1 + x_2 + \overline{x_3} + \overline{x_4}$$

would become

$$x_1 + x_2 + (1 - x_3) + (1 - x_4) \geq 1,$$

which is further turned into

$$x_1 + x_2 - x_3 - x_4 \geq -1.$$

Note that the right hand side of the inequality is 1 minus the number of negative literals in the row. Let c_j be the cost of setting x_j to 1. The ILP is therefore:

$$\min \sum_{j=1}^{m} c_j x_j, \text{ subject to: } \sum_{j=1}^{m} a_{ij} x_j \quad \geq \quad 1 - p_i, \ i = 1, ..., n \quad (1)$$

$$x_j \quad \in \quad \{0, 1\}, \ j = 1, ..., m \quad (2)$$

where a_{ij} is 1 if x_j appears in the true form in row i and -1 if it appears in the complement form, and p_i is the number of negative literals in row i.

The *linear-programming relaxation (LPR)* of an ILP is the linear program obtained by disregarding the integrality constraints. For the above ILP, the LPR is derived by simply replacing the 0-1 constraints (Constraint (2)) with $x_j \geq 0$. The *dual* of the LPR thus obtained is:

$$\max \sum_{i=1}^{n} (1 - p_i) y_i \quad (3)$$

$$\text{subject to: } \sum_{i=1}^{n} y_i a_{ij} \quad \leq \quad c_j, \ j = 1, ..., m \quad (4)$$

$$y_i \quad \geq \quad 0, \ i = 1, ..., n. \quad (5)$$

B. Lower Bounding

The traditional approach to solving an integer linear program is to first solve its LPR. If the optimal solution of the LPR consists of integers, then we have an optimal solution for the original ILP as well. Otherwise, we pick a variable (using various selection heuristics) and partition the original problem into two subproblems (the branching step). Suppose x_j is the selected variable, and x_j^* was the value of x_j in the optimal LPR solution. The two subproblems are derived from the original LPR by adding the constraints $x_j \geq \lceil x_j^* \rceil$ and $x_j \leq \lfloor x_j^* \rfloor$. We then proceed to solve one of these problems following the same steps, eventually reaching an integer solution (provided the problem is integer-feasible). We also attempt to solve the other problem. If at any point in the process we determine that the lower bound of a subproblem exceeds the objective value of the best (integer) solution found so far, then that subproblem can be pruned.

It is readily apparent that, for an integer linear program, the optimal value of the objective function for the corresponding linear program without the integrality constraints gives a lower bound for the ILP, because any feasible solution for the ILP is also a feasible solution for the LPR. This lower bound is typically used in ILP solvers. Various techniques have been proposed to compute lower bounds. One approach is called *dual ascent*, a greedy algorithm based on the weak duality theorem [9]: the objective function value of a dual *feasible* solution is less than or equal to that of any primal feasible solution. Dual ascent was used as the first step for solving unate problems in [2].

We now show that, for uniform-cost unate problems, dual ascent yields dual feasible solutions that are equivalent to independent sets. Recall that an independent set of rows in a covering problem is a subset of rows of which no two rows are covered by the same column. For instance, in the covering matrix shown in Figure 1, rows r_2 and r_4 form an independent set. A solution for a unate problem must cover at least the rows in an independent set, of which each row must be covered by at least a distinct column. Thus, for a unicost unate problem, the cardinality of an independent set gives a lower bound.

r_1	1	1	1	1	1										
r_2	1					1	1	1	1						
r_3		1				1				1	1	1			
r_4			1				1			1			1	1	1
r_5				1				1			1		1		1
r_6					1				1			1		1	1

Fig. 2. An ill-conditioned covering matrix (from [3]). Assuming uniform cost, the maximal independent set and dual ascent give a lower bound of 1, whereas LPR gives a lower bound of 3.

r_1	1	1	
r_2	1		1
r_3		1	1

Fig. 3. The optimal values of the LPR is 1.5, but we can round it up and use 2 as the lower bound.

Theorem 1: For a unicost unate covering problem, an integral solution to the dual problem of the LPR corresponds to an independent set.

Proof: Let y_i be the the dual variables associated with the LPR of the covering problem. Then each y_i must satisfy, for every j:

$$\sum_{i=1}^{n} y_i a_{ij} \quad \leq \quad 1. \quad (6)$$

Thus, every dual variable y_i must be less than or equal to 1. Since dual ascent works with integral values only, y_i is either 1 or 0. Furthermore, if there is a column j is such that a_{ij} and $a_{i'j}$ are both 1, then y_i and $y_{i'}$ must not be both set to 1; otherwise, Constraint (6) would be violated. Therefore, the rows for which the corresponding dual variables are set to 1 do not intersect and hence form an independent set. This shows that the quality of lower-bound estimates obtained by a dual ascent procedure is equivalent to that obtained by independent-set-based procedures. ■

Since many problems in our application domain have uniform or near-uniform cost functions, the dual ascent procedure is unlikely to provide better lower bounds. For covering problems with a wide-varying cost function, a dual ascent procedure may yield a better lower bound than independent set, because, in dual ascent, rows for which the dual variables are strictly positive may intersect. In any case, both independent set and dual ascent provide dual *feasible* solutions, the values of which are by definition less than or equal to that of a dual *optimal* solution. Thus, we propose that the linear-programming relaxation be solved with an LP-solver. Experimental results demonstrate that the optimal value of the LPR is almost always greater than the independent-set-based lower bound. Even though solving the LPR requires much more effort than the aforementioned heuristics for lower-bound estimation, we have found that it yields sufficiently tight estimates that the amount of time saved with the reduction of search space outweighs that expended by the additional effort.

To illustrate the superiority of LPR-based lower bounds, consider the example from [3], shown in Figure 2. Since every row intersects with every other row, the maximal independent set consists of one row only. Assuming every column has a uniform cost of 1, independent-set gives a lower bound of 1, as does dual ascent. However, the LPR of this covering problem has an optimal value of 3, which is clearly a tighter bound. Indeed, the optimal solution of the covering problem itself is 3.

C. Observations

We have observed the following which allow us to improve the efficiency of lower-bound computation:

1. It is clear that the *ceiling* of the optimal solution value of an LPR may be used as the lower-bound. A simple example is shown in Figure 3. The optimal value of the LPR is 1.5, but 2 is a valid lower bound for the integral solutions. However, it is necessary to be somewhat conservative in taking the ceiling in order to tolerate for rounding errors in floating-point computation. In our experiments we have observed that in some cases the optimal value of an LPR is integral, but due to rounding errors the LP solver gives a *slightly larger* value (on the order of 10^{-8}) than the true optimal value. Taking the ceiling of this value would then result in an incorrect lower bound. A

r_1	1	1				
r_2	1		1			
r_3		1	1			
r_4				1	1	
r_5				1		1
r_6					1	1

Fig. 4. This matrix can be decomposed into two, one consisting of rows r_1–r_3, the other r_4–r_6. The optimal value of the LPR for the entire matrix is 3. However, if we solve the LPRs of the two submatrices separately, we obtain 4 as the lower bound.

tolerance of $\epsilon = 0.005$ (i.e., using $\lceil z - \epsilon \rceil$ instead of $\lceil z \rceil$) is sufficient for all our benchmark problems.

2. If a covering matrix can be decomposed into independent subproblems, then we may solve the LPR of each subproblem independently, and add the ceilings (with numerical tolerance) of the optimal values to obtain a lower bound, instead of adding the optimal values first and then taking the ceiling. Consider the covering matrix in Figure 4, which consists of two independent submatrices. If we solve the LPR of the entire matrix, we obtain the optimal value of 3. However, if we solve the LPRs of the two submatrices separately, each has an optimal value of 1.5. We may first round up each, and obtain a better lower bound of $\lceil 1.5 \rceil + \lceil 1.5 \rceil = 4$. An additional benefit of decomposition is that the smaller LPRs can be solved more quickly.

3. Some linear programs are solved more quickly with the primal simplex procedure, while others are amenable to the dual simplex procedure. For this reason, most commercial solvers provide both methods. We have observed experimentally that if the LPR of the initial covering problem can be more quickly solved with the primal simplex procedure (respectively, dual), then it is likely to be more beneficial to use the primal (respectively, dual) procedure throughout the branch-and-bound process.

4. When solving the LPR using the dual simplex method, which maintains a dual feasible solution, we may terminate the search for the optimal solution to the LPR at any point and still obtain a valid lower bound (by the weak duality theorem). Thus, if we can quickly estimate an *upper bound* on the dual problem and decide that the current dual solution is good enough, we need not carry out the dual simplex procedure to completion.

One common case where this observation finds application is the following. Suppose the best integral solution found so far in the branch-and-bound process has a cost of b, which is an upper bound for the dual problem. Then, as soon as the objective function value of the dual solution reaches $b - 1 + \epsilon$, we may terminate the dual simplex procedure. If we do not have direct control over the LP solver, we may change the optimization problem D into a *feasibility* problem D'. That is, instead of optimizing for the dual objective (3), we check for the feasibility of the following linear program D':

$$\max 0, \text{ subject to: } \sum_{i=1}^{n} y_i a_{ij} \leq c_j, \; j = 1, ..., m \tag{7}$$

$$\sum_{i=1}^{n} (1 - p_i) y_i \geq b - 1 + \epsilon \tag{8}$$

$$y_i \geq 0, \; i = 1, ..., n. \tag{9}$$

Note that the objective function of D has now become Constraint (8) in D', and the objective function of D' is simply a constant since we are interested in feasibility only. If D' has a feasible solution, then the optimal value of D must be greater than or equal to $b - 1 + \epsilon$. This in turn means that the lower bound we set out to compute is at least b, and since this lower bound equals or exceeds the cost of the current best solution, we may terminate the present branch immediately. On the other hand, if D' does not have a feasible solution, then it is still possible that the present branch contains a better solution; hence, the search in this branch is continued. This method is particularly effective when the last steps of the dual procedure would consist of very small increments to the objective function value and these increments would not cross integral boundaries.

5. Given the optimal solution of an LP problem X, a simplex-based LP solver can very quickly find the optimal solution of another problem that is incrementally different from X, provided the impact of degeneracy is small. Thus in the branching step, when we select or reject a column, we may set the corresponding variable to 1 or 0 and obtain a potentially better lower bound. However, if the selection or rejection of a column results in a large reduction in the size of the covering matrix, it is typically better to start the LPR afresh with the reduced matrix. This is an important observation that significantly affects the quality of results.

D. The Covering Procedure

Our solver, Binate Covering Ultra (BCU), consists of the following steps:

[Initial] Initially, matrix reductions based on essentiality, dominance, Gimpel's technique, and decomposition into independent submatrices are applied. The LPR of the covering matrix is generated, and both the primal and dual procedures are applied and computation times measured. The more efficient of the procedures chosen for lower-bound estimation throughout the remainder of the branch-and-bound process.

[Integral] If at any point the solution of the LPR is integral, then we have a solution for the current branch; otherwise, continue with the branching step (**Branch** below).

[Branch] Choose a column according to a heuristic measure derived from the cost of the column, the number of rows it covers, and the minimum cost of covering each of these rows. These criteria are similar to [12]. Apply the reduction again. Estimate the lower bound of the reduced matrix using LPR. This estimation of the lower bound is done incrementally, unless the matrix has been reduced significantly from the previous level of recursion. If the lower bound exceeds the cost of the best solution found so far, prune this branch. Otherwise, continue **Branch**.

[Other branch] Reject the column previously chosen and continue the branch-and-bound process as in **Branch**. Pick the better solution of the two branches.

IV. Experiments and Results

We present two sets of results. The first set of results, summarized in Table I, provides a comparison between the lower bounds obtained using the maximal independent set (MIS) and LPR methods. The table compares the *first* bound obtained by these methods, on the original problem. As can be seen the LPR lower bounds are superior (larger) in virtually every case. Note that even a slightly better bound can result in exponentially better performance, due to the pruning of nodes in the search tree. Of course, the bound has to be improved as variables are set to particular values. We have found that, in general, if the LPR bound is superior to the MIS bound (and it almost always is) for the original problem, it remains superior through the branching search as variables are set to particular values, up until the subproblems become trivial.

We next provide comparisons between Scherzo [3], our solver BCU, and a commercial LP/ILP solver widely recognized as being one of the best (which we also used to compute our lower bounds). These results are summarized in Table II. All three programs were run on the same machine, on a 125MHz HyperSparc 20 with 128MB of memory.

BCU significantly outperforms Scherzo on large examples such as **prom2**, **ex5** and **apex7.b**, as well as many others. BCU also outperforms the commercial solver in many cases, and significantly so in some examples such as **des.a**, **apex4.a**, and **apex7.b**. For the example **alu4.b**, which none of the three programs can prove optimality within the allotted time, BCU discovers a better solution than either Scherzo or the commercial solver.

In the cases where Scherzo is superior by a large factor, for example, **des.a** and **C880.b**, the improved performance is not because of better bounding, but because of two factors. Scherzo implements caching strategies that "remember" previous subproblems and reuse previously computed solutions, and different variable selection heuristics can result in smaller branching trees. Caching strategies can be implemented within the BCU framework as well, and are expected to improve the efficiency of the solver.

The commercial solver performs better than Scherzo in many examples. The main reason why the commercial solver does better than BCU in some of the examples is because the commercial solver has access to core linear programming routines, which we do not currently have. We elaborate on this in the next section.

V. Summary and Future Work

We have presented an efficient approach to solving unate and binate covering problems. Our approach is based on the estimation of lower bounds using linear-programming relaxation. Although LPR has been extensively used in solving integer linear programming problems, its power has been largely overlooked in solving covering problems. Our experimental results show that LPR indeed yields much better lower bounds than previously widely used methods based on independent sets. On the other hand, the traditional reduction techniques for covering problems are also necessary in order to exploit properties specific to covering problems—instead of solving the covering problem as just an ILP. Further, while LPR bounds can be expensive to compute, they can be computed incrementally during the branching search to increase efficiency. The combination of these techniques results in many improvements: not only were we able to handle previously-solved problems more efficiently, but also in some cases we were able to finish on difficult problems that other solvers could not complete, or to find a better solution in the same amount of time. We conclude that, in light of the presented experimental results, future research should focus on speeding up LPR-based lower bounding, and use independent-set approaches only for pruning techniques such as Coudert's limit-lower bound [3].

In addition to lower bound computation, variable selection methods have a significant impact on the efficiency of the solver. In BCU we have used a single uniform variable selection heuristic for all examples: nonintegral variables were picked according to the criterion described in the **Branch** step of Section III-D. Enhancing the variable selection method may result in faster execution, and we are currently investigating methods that incorporate information from the solution of the LPR.

We have used a commercial LP solver as a "black box," i.e., we used it to solve the LPRs without taking advantage of information available in intermediate steps. If we had access to information such as degeneracy and a tight upper bound, and if we had control over the progress of the solver, we might be able to speed up the lower-bound estimation substantially. For instance, various pieces of information gathered from intermediate steps may provide useful heuristic guidance for determining when and how often to use LPR. We believe that the tight integration of the branch-and-bound and the LPR procedure is the key to further improvements.

Acknowledgements

The authors are grateful to Rick Rudell for providing his binate-covering solver, in whose framework the algorithm described in this paper was implemented; and to Olivier Coudert for providing the benchmark results of his solver *Scherzo*. The authors also thank the reviewers for their valuable suggestions.

References

[1] P. Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-boolean optimization. Technical Report MPI-I-95-2-003, Max-Planck-Institut für Informatik, January 1995. See *www.mpi-sb.mpg.de/guide/staff/barth/publications/reports.*

[2] J. E. Beasley. An algorithm for set covering problem. *European Journal of Operational Research*, 31:85–93, 1987.

[3] O. Coudert. On solving binate covering problems. In *Proceedings of ACM/IEEE Design Automation Conference*, June 1996.

[4] O. Coudert and J.-C. Madre. New ideas for solving covering problems. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 641–646, June 1995.

[5] J. F. Gimpel. A reduction technique for prime implicant tables. *IEEE Trans. on Elec. Comp.*, 14:535–541, June 1965.

[6] A. Grasselli and F. Luccio. A method for minimizing the number of internal states in incompletely specified machines. *IEEE Trans. on Elec. Comp.*, 14:350–359, June 1965.

[7] S. Liao, S. Devadas, K. Keutzer, and S. Tjiang. Instruction selection using binate covering for code size minimization. In *Proceedings of 1996 International Conference on Computer-Aided Design*, pages 393–399, November 1996.

[8] E. L. McCluskey, Jr. Minimization of boolean functions. *Bell Sys. Tech. Journal*, 35:1417–1444, April 1959.

[9] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, 1988.

[10] W. V. O. Quine. A way to simplify truth functions. *Am. Math. Monthly*, 62:627–631, 1955.

[11] W. V. O. Quine. On cores and prime implicants of truth functions. *Am. Math. Monthly*, 66:755–760, 1959.

[12] R. L. Rudell. *Logic Synthesis for VLSI Design*. PhD thesis, UC Berkeley, 1989. UCB/ERL M89/49.

Example				Lower Bound	
Name	row	× col	sol	MIS	LPR
lin.rom	545	× 578	113	109	113
test1	271	× 481	110	90	94
foutL	152	× 427	364	333	364
m4L	392	× 621	1335	1260	1331
mlp4L	418	× 587	1316	1279	1314
max512L	419	× 514	1087	1039	1087
ocex1	328	× 567	69	62	68
ocex2	575	× 681	162	154	161
max1024	917	× 904	259	197	208
prom2	1525	× 1813	287	248	260
ex5	687	× 974	37	31	36
add4	883	× 858	31	27	31
5xp1.b	859	× 838	12	9	11
count.b	694	× 913	24	17	24
e64.a	461	× 492	80	65	80
sao2.b	779	× 695	25	24	25
jac3	1254	× 266	15	12	15
des.a	17920	× 12183	942	822	918
int13	2439	× 516	110	98	109
apex4.a	11912	× 8406	776	722	774
C880.b	1859	× 1460	63	54	62
int10	32362	× 1376	116	103	115
int14	3479	× 618	140	115	135
apex7.b	2353	× 2126	37	29	35
alu4.b	1838	× 1492	—	40	47

TABLE I

Characteristics of examples. The second column shows the sizes of examples. The column *sol* gives the optimal solution of the covering problem, if known. The columns *MIS* and *LPR* exhibit the first lower bound obtained by maximal independent set and by linear programming relaxation, respectively.

Example		Scherzo 96		BCU		ILP solver	
Name	sol	UB	CPU	UB	CPU	UB	CPU
lin.rom	113	113	1.3	113	27.7	113	1.8
test1	110	110	4.4	110	2.9	110	0.8
foutL	364	364	44.8	364	2.4	364	0.2
m4L	1335	1335	16.7	1335	4.6	1335	1.5
mlp4L	1316	1316	10.3	1316	2.5	1316	0.9
max512L	1087	1087	94.2	1087	0.6	1087	0.8
ocex1	69	69	164.1	69	18.6	69	20.7
ocex2	162	162	117.0	162	31.4	162	3.9
max1024	259	259	8951.0	259	1886.7	259	2537.7
prom2	287	287	10211.1	287	918.0	287	138.0
ex5	65	65	14969.7	65	439.4	65	458.7
add4	31	31	6.1	31	1.9	31	1.7
5xp1.b	12	12	7.9	12	8.1	12	165.8
count.b	24	24	55.3	24	0.9	24	12.6
e64.a	80	80	0.5	80	0.7	80	0.3
sao2.b	25	25	1.0	25	94.1	25	33.1
jac3	15	15	5.9	15	4.6	15	5.0
des.a	942	942	125.2	942	1151.5	974	>20000
int13	110	110	126.0	110	3.3	110	2.2
apex4.a	776	776	127.4	776	23.3	793	>20000
C880.b	63	63	200.7	63	3347.8	63	2140.0
int10	116	116	433.7	116	262.9	116	35.1
int14	140	140	5627.6	140	61.9	140	15.4
apex7.b	37	38	>20000	37	1230.4	38	>20000
alu4.b	—	51	>20000	50	>20000	51	>20000

TABLE II

Comparison of Scherzo 96 [3], our solver BCU, and a commercial ILP solver. The columns *UB* give the best solutions found by each one of the solvers. CPU times are measured in seconds, on a 125MHz HyperSparc 20.