RUTCOR
RESEARCH
REPORT

A Branch-and-Bound Algorithm for a Family of Pseudo-Boolean Optimization Problems

Tibérius O. Bonates[a] Peter L. Hammer[b]

RRR 21-2007, July 3, 2007

RUTCOR
Rutgers Center for
Operations Research
Rutgers University
640 Bartholomew Road
Piscataway, New Jersey
08854-8003
Telephone: 732-445-3804
Telefax: 732-445-5472
Email: rrr@rutcor.rutgers.edu
http://rutcor.rutgers.edu/∼rrr

[a]RUTCOR – Rutgers Center for Operations Research, 640 Bartholomew Road, Piscataway, NJ 08854
[b]RUTCOR – Rutgers Center for Operations Research, 640 Bartholomew Road, Piscataway, NJ 08854

Rutcor Research Report
RRR 21-2007, July 3, 2007

A Branch-and-Bound Algorithm for a Family of Pseudo-Boolean Optimization Problems

Tibérius O. Bonates Peter L. Hammer

Abstract. Let $\Omega \subset \{0,1\}^n$ be a set of binary vectors, and let us associate to each vector $\omega \in \Omega$ a real weight. Consider now the problem of finding a conjunction C such that the sum of the weights of the vectors in Ω satisfying C is maximized. This problem can be formulated as a pseudo-Boolean optimization problem in which every term has degree n. This problem is a generalization of the so-called "maximum pattern problem" and has a natural application in an iterative algorithm for regression with binary predictors. We propose a simple branch-and-bound algorithm for this class of pseudo-Boolean problems and analyze its performance on a number of randomly generated instances.

Acknowledgements: We are deeply thankful to Alex Kogan, for helpful discussions on a preliminary version of this text. The first author would like to acknowledge the generous support of a DIMACS Graduate Student Award.

1 Introduction

In this report we describe a branch-and-bound algorithm for a family of pseudo-Boolean optimization problems, and report the results of a series of computational experiments that compare the solution quality and running time of our algorithm with those of the Xpress [17] integer linear programming solver.

2 Related Work

The branch-and-bound approach dates back to 1960 when the pioneer work of Land and Doig [29] was published, in which they described a novel algorithm for solving integer linear programming problems. Since then, several versions of the algorithm were proposed for special classes of optimization problems, such as the traveling salesman problem [20, 35, 38], facility location [19, 32, 43], network design [16, 21, 27], nonlinear programming problems [2, 25, 44, 46], to name a few.

The algorithm is essentially a combination of two procedures: the partitioning of the original problem into a series of successively simpler subproblems (*branching*), and the computation of bounds that permit the elimination of a significant fraction of the subproblems (*bounding*). The expectation is that large parts of the search space can be removed from consideration and only a small number of solutions have to be actually computed.

The branching procedure basically partitions the space of feasible solutions. The typical branching operation consists of (i) enumerating (or implicitly describing) the possible values of one or more variables that can actually be realized in a feasible solution, and (ii) partitioning the space of feasible solutions according to those values. Each partition is called a *branch* and creates a restricted version of the original problem. Therefore, the objective function value in each branch can be at most as good as that in the original problem. Note, however, that branching operations as described here do not remove any feasible solution from consideration. Thus, after a branching operation the set of solutions of a problem is completely preserved in its subproblems.

Due to the successively more refined partitions of the search space generated by the algorithm, we often refer to the *search tree* or the *branch-and-bound tree* associated to the application of a branch-and-bound algorithm to a particular instance. At the root of the tree we have a special node corresponding the original problem. Each of the remaining nodes of such a tree corresponds to a version of the original problem whose space of feasible solutions has been reduced by a branching operation. Two nodes are connected by an arc if one of them was obtained from the other by the application of a branching operation, i.e., one of them is a restricted version of the other. We usually say that the restricted subproblem is a *descendant* or a *child* node of the other, which is referred to as the *parent* node. In this paper we will sometimes identify a node in the search tree with the subproblem associated to it.

The bounding procedure relies on the computation of an explicit bound on the optimal objective function value for a given problem. Let us assume that our original problem is a

maximization one, and let z_f be the objective function value associated to a given feasible solution. We can use an upper bound on the optimal objective function value of a subproblem to conclude that it cannot contain an optimal solution to the original problem. If the upper bound on the optimal value of a given subproblem is $\hat{z}$, and $\hat{z} < z_f$, then it is clear that the given subproblem cannot contain an optimal solution to the original problem.

In most applications, the bounding procedure produces a solution for a relaxation of the subproblem, and that can be used as a candidate solution for the original problem. For example, in the case of integer linear programming problems, solving the linear relaxation of the current problem S provides a bound on the objective value of S while producing a candidate solution x for the original problem. If x turns out to be integral and the objective function value associated to it is better than that of the best solution of the original problem produced so far, we declare x the best known solution so far. If x is fractional, problem S can be either discarded or further branched on, depending on the value of the bound computed: if the objective function value associated to x is better than that of the current best known solution, then problem S is still considered for further branching; otherwise, x is discarded, and the current subproblem is removed from the search tree (as discussed above) without producing any descendant node. If the procedure for computing a bound on the optimal objective value of a subproblem does not produce a candidate solution, some mechanism of producing one or more candidate solutions from each subproblem is needed as a way of generating feasible solutions for the original problem.

More recently many modifications of the original branch-and-bound algorithm have been proposed based on combinations of the original principles of branching and bounding with those of other techniques, such as primal heuristics [3, 5, 34], cutting planes [20, 26, 35, 36, 38, 46], column generation [4, 5, 41, 45], and constraint satisfaction [1, 31, 37, 47]. These variants are mainly used for solving specific classes of problems, as they typically explore the structure of the problem being solved, for instance by the use of valid inequalities and special-purpose heuristics. For the solution of real-life problems, commercial solvers rely heavily on branch-and-bound algorithms, implementing some version of the standard algorithm enriched by a number of preprocessing techniques, heuristics for fixing of variables, branching and node selection strategies [1, 17, 26, 42], and general-purpose cutting planes [39, 40, 14].

Certain classes of combinatorial optimization problems possess structural properties that allow the opportunistic use of branch-and-bound algorithms. In this paper we describe a class of pseudo-Boolean optimization problems [9, 10, 15, 22, 23, 24] with applications in Logical Analysis of Data, and present a simple branch-and-bound algorithm that satisfactorily solves problems from this class.

In the next section we describe the pseudo-Boolean optimization problem considered in this paper. Section 4 presents the branching strategies of our algorithm, and Section 5 describes the node selection strategy utilized. In Section 6 we discuss the bounding procedure, and in Section 7 we report on computational experiments carried out on randomly generated instances. Finally, Section 8 discusses the contributions of the paper.

3 Problem Formulation

Let us consider a set of binary points $\Omega \subset \{0,1\}^n$, with $|\Omega| = m$, and let us associate to each point $\omega^k \in \Omega$ a real weight β_k. Consider now the problem of finding a conjunction C such that the sum of the weights of the points in Ω satisfying C is maximized. The following pseudo-Boolean optimization problem asks for such a conjunction:

$$\begin{aligned} \text{(PB)} \qquad \text{maximize} \qquad & \phi = \sum_{k=1}^{m} \beta_k \left(\prod_{i:\omega_i^k=0} \overline{y_i} \prod_{j:\omega_j^k=1} \overline{z_j} \right) \\ \text{subject to:} \qquad & y_j, z_j \in \{0,1\},\ j = 1,\ldots,n, \end{aligned}$$

where y_j is a Boolean variable associated to the inclusion of literal x_j in the resulting conjunction C, and z_j is a Boolean variable representing the inclusion of literal $\overline{x_j}$ in C, for $j = 1,\ldots,n$. Note that the weights β_k $(k = 1,\ldots,m)$ do not have necessarily the same sign.

Problem (PB) describes a particular class of pseudo-Boolean optimization problems in which every term has degree equal to n and variables appear only complemented in the expression of ϕ. As a simple example, let $n = 3$, $\Omega = \{(1,0,1),(1,1,0)\}$, $\beta_1 = 1$ and $\beta_2 = -1$. Then, function ϕ equals to $\overline{z_1}\,\overline{y_2}\,\overline{z_3} - \overline{z_1}\,\overline{z_2}\,\overline{y_3}$.

We remark here that the so-called *Maximum Pattern Problem* described in [8] is a special case of problem (PB) if the fuzziness parameter is set to zero. In such a case, we assume that $\Omega = \Omega^+ \cup \Omega^-$, with $\Omega^+ \cap \Omega^- = \emptyset$, and we ask, for instance, for a conjunction that covers the largest number of points from Ω^+ and does not cover any of the points in Ω^-. This is equivalent to setting weights $\beta_j = 1$ for the points $\omega^j \in \Omega^+$, $\beta_j = -|\Omega^+|$ for the points $\omega^j \in \Omega^-$, and solving problem (PB).

Problem (PB) can also be used to produce maximum patterns with nonzero fuzziness by a proper setting of the weights β_j. In that case, however, (PB) is not equivalent to the formulation described in [8], since we cannot impose a nontrivial strict upper bound on the number of points from Ω^- that are covered. On the other hand, by adjusting the weights β_j we can obtain fuzzy patterns with optimal coverage with respect to a criterion closer to the one used in the context of subgroup discovery [30].

A more natural application of problem (PB) is in the context of pseudo-Boolean regression [6, 7], in which (PB) arises in the subproblem phase of a column generation algorithm. In that specific application, each point is assigned a real weight and the goal is to find a conjunction whose weighted coverage is positive (corresponding to a positive reduced cost within the column generation context), rather than finding a conjunction with the largest possible weighted coverage.

4 Branching Strategy

A simple branch-and-bound algorithm for a problem with binary variables fixes one variable at each branch. In such a case, every subproblem in the branch-and-bound algorithm produces exactly two descendants, one corresponding to fixing $x_j = 0$ and the other to fixing

$x_j = 1$. The enumeration associated to such a branching policy is typically very large, causing such an algorithm to be of little practical use. In general, if a simple branching strategy is to be used, it becomes necessary to use preprocessing techniques that substantially simplify the problem (reducing the number of variables or the space of feasible solutions) and to develop sharp bounds to prune large parts of the search tree.

Some classes of problems display certain structural properties that allow the development of specific branching strategies that fix a larger number of variables at each branch of the search tree, quickly reducing the size of the subproblems during the execution of the algorithm. In this section we describe a branching strategy that takes advantage of the particular structure of problem (PB) and attempts to fix a large number of variables at each branch.

We propose a branching rule that takes into account the large number of variables involved in the clauses (or terms) in problem (PB). We perform the branching step on an entire clause of ϕ, instead of on a single variable. Branching on a clause $T = \prod_{i \in A} \overline{y_i} \prod_{j \in B} \overline{z_j}$ can fix a large number of variables at once, significantly reducing the sizes of the resulting subproblems. For instance, if we fix clause T at value 1, we simultaneously fix all variables $y_i = 0$ ($i \in A$) and $z_j = 0$ ($j \in B$). Fixing $T = 0$ leaves us with a number of options: fixing any of the variables involved in T at value 1 is enough to ensure that $T = 0$. We can then create $|A| + |B|$ descendant nodes associated with the case $T = 0$, with an increasing number of variables fixed over these nodes. To achieve that, let us assume some order among the variables involved in T. We can fix the first variable at the value 1, leaving the remaining variables free. As a separate branch we can fix the first variable at the value 0 and the second variable at the value 1, with the remaining variables being free. Following this process, the third variable can be fixed at 1, and the first two at 0, and so on, until we have a descendant node that fixes the last variable at 1 and all other variables in T at 0. Clearly, the set of nodes described above contains all possible assignments with $T = 0$, and every such assignment is implicitly defined by exactly one of the nodes in the set.

Consider the example of $T = \overline{y_1}\,\overline{z_2}\,\overline{z_3}\,\overline{y_4}$. Branching on $T = 1$ fixes $y_1 = z_2 = z_3 = y_4 = 0$ simultaneously and reduces the number of variables in the problem by 4. Fixing $T = 0$ can be done in four different ways: fixing each variable in T at the value 1 separately. If we simply fix each variable separately at the value 1, we have a symmetry effect that can clearly create an overhead in the performance of the algorithm. For example, as we fix $y_1 = 1$, and in a separate branch fix $z_2 = 1$, we are potentially considering a number of identical subproblems in subsequent branches: at subsequent rounds of branching we could obtain $(y_1 = 1, z_2 = 1)$ in one part of the search tree and $(z_2 = 1, y_1 = 1)$ in another part. Since the number of possible solutions is exponential, this symmetry effect can result in a large increase in the running time of the algorithm. In order to prevent such a situation we fix an increasingly larger set of variables at each branch by, say, fixing $y_1 = 1$, and in a subsequent branch fixing simultaneously $y_1 = 0$ and $z_2 = 1$. In a third branch we can fix $y_1 = z_2 = 0$ and $z_3 = 1$, and finally fix $y_1 = z_2 = z_3 = 0$ and $y_4 = 1$ in the last branch. Note that this policy does not remove any potential solution from consideration, while preventing the undesirable overhead of evaluating equivalent subproblems multiple times.

4.1 Branching Criterion

Let us assume that ϕ has both terms with positive and with negative coefficients. Indeed, if all weights β are positive, the optimal solution of (PB) is simply the empty pattern (i.e., a conjunction containing no literals, covering all points in Ω), since problem (PB) requires the maximization of ϕ. Similarly, if all points have negative weights β, then any conjunction that is not satisfied by any point in Ω is an optimal solution of (PB). For instance, any conjunction of the type $x_j\overline{x_j}$ (corresponding to $y_j = z_j = 1$ in (PB)) is an optimal solution in this case.

A simple criterion for selecting the next clause to branch on is the value of the coefficient of the clauses. The clause T with largest positive coefficient will hopefully provide a large increase in objective function when fixed to 1. Thus, we first branch on the case $T = 1$, postponing the branches with $T = 0$.

Similarly, if in a certain node there are only negative clauses, we choose the most negative clause and evaluate first the branches with $T = 0$, and subsequently the one with $T = 1$. The advantages of such a criterion are that its implementation is straightforward and its complexity is linear on the number clauses at the given node.

Obviously, more sophisticated criteria can be used, which can potentially reduce the depth of the search tree and provide sharper upper bounds. In particular, the selection of the order in which the variables should be fixed in the branches with $T = 0$ can make a difference, since fixing certain variables at the value 1 can potentially cause other clauses to vanish.

In addition to that, one could take into account the total effect of fixing a certain clause to the value 1. Fixing $T = 1$ can simultaneously fix other clauses at the value 1 (those involving only a subset of the literals in T), thereby affecting the total contribution of that fixing to the objective function.

In our computational experiments with randomly generated problems the use of the alternative criterion described above did not result in an overall improvement of the algorithm, as compared to the use of the simple criterion. In some cases, marginally superior solutions were obtained early in the search, but that behavior was not consistent. Moreover, in all cases the running time of the algorithm was largely increased, sometimes to twice as much as the time required with the simple branching criterion. Indeed, it is clear that the complexity of the more sophisticated criterion is significantly higher.

In the remainder of this paper we utilize solely the simple criterion proposed in the beginning of this section.

5 Initialization and Node Selection

In this section, we describe a simple heuristic for finding initial solutions of good quality, and the node selection strategy used throughout the rest of the algorithm.

5.1 Generating an Initial Solution

As described above, the bounding principle depends on the objective function value of the best solution found so far. Near-optimal solutions typically allow the pruning of considerable parts of the search tree, while solutions of poor quality render the bounding procedure ineffective [5, 18, 28, 33]. Since at early stages of the search the space of solutions is still very large, finding a solution with a high objective function value at the outset of the search contributes to an aggressive pruning that can reduce the overall running time of the algorithm.

We utilize the greedy branching strategy described in the previous section, along with a depth-first queue discipline in order to produce one or more solutions of good quality at an early stage of the search process. We select the next clause to be branched on, and follow the appropriate branch, postponing the other branches. This policy is applied until a maximum number of K nodes is visited, where K was selected to be either 500 or 1,000 in our computational experiments.

As long as there are unresolved clauses, i.e., clauses that have not yet been fixed, the algorithm branches on a clause whose fixing seems to be the most advantageous one – according to the branching criterion described in the previous section – among all unresolved clauses.

This depth-first strategy for selecting the next node to be examined is extremely fast since it does not require the consideration of which node should be examined next based on its upper bound. The procedure usually produces solutions of good quality after a relatively small number of branching operations.

For problems of small dimension, the procedure described above for generating an initial solution frequently exhausts the search space. For larger problems, the solutions produced in this initial phase of the algorithm are typically among the best solutions found during the remainder of the search.

In Section 7 we present the results of a series of computational experiments that support these claims.

5.2 Node Selection Strategy

We discuss below how the general node selection strategy used throughout the algorithm differs from the one used during the initialization phase.

At any stage of the search procedure, there is a set of nodes that have not yet been examined and whose corresponding upper bounds are still larger than the objective function value of the best solution found so far. We refer to these nodes as *open nodes*. A careful choice of the next open node to be processed can lead to substantial gains in computing time as solutions of good quality may be found more quickly, helping to further prune the search tree.

We select the best open node to be processed according to a discipline based on the depth-first strategy described for obtaining initial solutions. The main difference consists in the periodic interruption of the search every time a certain number of nodes has been examined. This interruption of the branch-and-bound search has the purpose of reorganizing

the memory used by the algorithm. Two main tasks are accomplished during each interruption: (i) open nodes, whose upper bounds have become worse than the objective function value of the best known solution since the time they were created, are discarded, and (ii) the remaining set of open nodes is sorted in decreasing order of their upper bounds, with the intent of giving priority of processing to nodes that are more likely to contain improved solutions. We apply such an interruption every time the number of nodes processed equals a multiple of an integer number, typically chosen between 100 and 2,000.

The strategy described above for selecting the next clause to branch on, and for deciding the order in which to create descendants of a subproblem is used throughout the entire search procedure.

6 Bounding

As discussed before, the combined use of an effective bounding procedure with feasible solutions of good quality is a crucial part of the branch-and-bound algorithm. At the same time, a good bounding procedure should be relatively fast. Indeed, the node selection strategy described above relies on the explicit computation of an upper bound for each open node in the search tree.

We utilize as an upper bound on the objective function value at any given node, having an associated function ϕ, the sum of the coefficients of the clauses of ϕ having positive coefficients, plus the independent term of ϕ. This is an optimistic upper bound, since we are expecting that there is a certain assignment of the remaining variables for which all clauses with a negative coefficient will vanish, while all clauses with a positive coefficient will be satisfied.

This rather simplistic upper bounding procedure is extremely inexpensive and turned out to work remarkably well in our computational experiments described in Section 7 below.

7 Computational Experience

In this section we report the results of a number of computational experiments with our branch-and-bound algorithm. We compare the performance of our algorithm with that of the commercial solver Xpress-MP [17] in terms of running time and quality of solutions produced. We also report the quality of the upper bounds generated by both algorithms. Our experiments were performed on randomly generated instances of problem (PB).

In our computational experiments, we report the performance of what we call "truncated" branch-and-bound searches, in which explicit limits are imposed on the maximum running time, or on the maximum number of branch-and-bound nodes evaluated, or on both.

We execute three types of experiments: a truncated search with 1,000 nodes, a truncated search with 2,000 nodes, and a truncated search without a limit on the maximum number of branch-and-bound nodes evaluated. For each type of experiment we established the maximum limit of 30 minutes of running time. It turned out that most of the small instances

were solved to optimality by both our algorithm and the Xpress-MP solver during one of the truncated searches. Neither our algorithm nor the Xpress-MP solver were able to solve the larger instances within the truncated searches.

Our algorithm was implemented in *C++* using the *MS Visual C++ .NET 1.1* environment, utilizing version 16.10.02 of the Xpress-MP solver in our experiments. The computer utilized to run all experiments reported in this paper was an *Intel Pentium* 4, 3.4GHz, with 2GB of RAM.

Table 1 presents the number of variables and the number of clauses of the set of randomly generated instances used in our experiments. Note that n is the number of variables in the set Ω. However, in problem (PB) the total number of binary variables is in fact equal to $2n$, while the total number of clauses equals m, each clause having a coefficient randomly chosen between -1 and 1.

Instance	n	m
p_20_1	20	30
p_20_2	20	30
p_20_3	20	30
p_30_1	30	50
p_30_2	30	50
p_30_3	30	50
p_40_1	40	70
p_40_2	40	70
p_40_3	40	70
p_50_1	50	100
p_50_2	50	100
p_50_3	50	100

Instance	n	m
p_70_1	70	120
p_70_2	70	120
p_70_3	70	120
p_100_1	100	150
p_100_2	100	150
p_100_3	100	150
p_150_1	150	200
p_150_2	150	200
p_150_3	150	200
p_200_1	200	250
p_200_2	200	250
p_200_3	200	250

Instance	n	m
p_250_1	250	300
p_250_2	250	300
p_250_3	250	300
p_300_1	300	400
p_300_2	300	400
p_300_3	300	400

Table 1: Characteristics of randomly generated instances of problem (PB): n = number of components of set Ω, m = number of points in set Ω.

Table 2 shows the results of a truncated search using a maximum of 1,000 branch-and-bound nodes for each problem. The columns labeled "Xpress-MP" refer to the solution and the running time required by the Xpress-MP solver to perform the truncated search. "BBPBF" stands for our "branch-and-bound algorithm for maximizing a pseudo-Boolean function." Within the limit of 1,000 branch-and-bound nodes, BBPBF found solutions with objective function values that are 37% larger on average than those found by the Xpress-MP solver, while requiring an average of 26% of the running time of the Xpress-MP solver.

Table 3 shows the results of a truncated search using a maximum of 2,000 branch-and-bound nodes. BBPBF clearly outperforms Xpress-MP in the vast majority of the instances. Not only it finds solutions that are on average 39% superior to those of Xpress-MP, but the running time required is 35% (on average) of the one required by Xpress-MP.

Table 4 presents the final upper bounds obtained by BBPBF and Xpress-MP after the end of the truncated searches of 1,000 and 2,000 nodes. Clearly, the upper bound computed

by BBPBF is sharper than that computed by the standard integer programming solver of Xpress-MP.

Tables 5 and 6 presents the results of a truncated search in which no limit is imposed on the number of branch-and-bound nodes, but a limit of 30 minutes is enforced on the running time of each algorithm. We report the objective function value of the best solution found during the search by both the BBPBF and Xpress-MP algorithms, along with the total running time and the number of nodes examined during the search.

8 Conclusions

It is easy to see that the BBPBF algorithm outperforms the Xpress-MP solver in each of the criteria evaluated in this study. While this might come as a surprise given that Xpress-MP is a well-established integer linear programming solver, we believe that it simply reflects our use of a priori information about the problem's structure.

In particular, it is natural that our bounding procedure proved more effective than the one provided by the continuous relaxation of the integer linear formulation of the problem. Moreover, our algorithm works on the original space of problem (PB), while the use of the Xpress-MP solver requires the rewriting of the original problem as an integer linear program, with as many additional binary decision variables as the number of clauses, and several additional constraints.

It is possible that a proper fine tuning of the cutting-plane regime and preprocessing techniques of the Xpress-MP solver could prove useful in speeding up the solution of (PB) instances. However, the consideration of a fine tuning of the parameters of both algorithms would be part of a larger study and is beyond of the scope of this paper. Moreover, as discussed in Section 3, one of the contexts in which problem (PB) arises naturally is in that of pseudo-Boolean regression, where one is interested in producing solutions of reasonable quality, under low computational cost, rather than proving optimality.

Although we have focused our description of the algorithm and our computational experiments on problems of the form (PB), it is clear that the BBPBF algorithm can be applied to arbitrary unconstrained pseudo-Boolean optimization problems, without any restriction on the degree of the terms. It is likely that for problems with small degree terms, such as quadratic unconstrained binary optimization (QUBO) [11, 12, 13], the performance advantage of our algorithm will be diminished, as compared to algorithms such as the Xpress solver or a standard branch-and-bound algorithm that branches on individual variables. For problems having a relatively large average degree of terms, it is more likely that the BBPBF algorithm will still retain a significant advantage over the above mentioned algorithms.

Finally, another desirable feature of our algorithm is its trivial portability to different computer platforms, since it is a standard stand-alone *C++* implementation that does not rely on the use of any third-party solver or library.

References

[1] Achterberg T. SCIP-a framework to integrate constraint and mixed integer programming. *Konrad-Zuse-Zentrum fur Informationstechnik Berlin, ZIB-Report*, pages 04–19, 2004.

[2] Ahmed S., M. Tawarmalani, N.V. Sahinidis. A finite branch-and-bound algorithm for two-stage stochastic integer programs. *Mathematical Programming*, 100(2):355–377, 2004.

[3] Balas E., M.J. Saltzman. An Algorithm for the Three-Index Assignment Problem. *Operations Research*, 39(1):150–161, 1991.

[4] Barnhart C., C.A. Hane, P.H. Vance. Using Branch-and-Price-and-Cut to Solve Origin-Destination Integer Multicommodity Flow Problems. *Operations Research*, 48(2):318–326, 2000.

[5] Barnhart C., E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, P.H. Vance. Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, 46(3):316–329, 1998.

[6] Bonates T.O., P.L. Hammer. Pseudo-Boolean Regression. *Proceedings of the XIX International Symposium on Mathematical Programming*, 2006.

[7] Bonates T.O., P.L. Hammer. Pseudo-Boolean Regression. Technical Report RRR 3-2007, RUTCOR – Rutgers Center for Operations Research, 2007.

[8] Bonates T.O., P.L. Hammer, A. Kogan. Maximum Patterns in Datasets. *Discrete Applied Mathematics, In Press (doi: 10.1016/j.dam.2007.06.004)*, 2007.

[9] Boros E., P.L. Hammer. Cut-polytopes, Boolean quadratic polytopes and nonnegative quadratic pseudo-Boolean functions. *Mathematics of Operations Research*, 18(1):245–253, 1993.

[10] Boros E., P.L. Hammer. Pseudo-Boolean optimization. *Discrete Applied Mathematics*, 123(1-3):155–225, 2002.

[11] Boros E., P.L. Hammer, G. Tavares. Local search heuristics for unconstrained quadratic binary optimization. Technical Report RRR 09-2005, RUTCOR - Rutgers Center for Operations Research, Rutgers University, 2005.

[12] Boros E., P.L. Hammer, G. Tavares. A max-flow approach to improved lower bounds for quadratic 0-1 minimization. Technical Report RRR 07-2006, RUTCOR - Rutgers Center for Operations Research, Rutgers University, 2006.

[13] Boros E., P.L. Hammer, G. Tavares. Preprocessing of unconstrained quadratic binary optimization. Technical Report RRR 10-2006, RUTCOR - Rutgers Center for Operations Research, Rutgers University, 2006.

[14] Chvátal V. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305–337, 1973.

[15] Crama Y., P. Hansen, B. Jaumard. The basic algorithm for pseudo-Boolean programming revisited. *Discrete Applied Mathematics*, 29(2-3):171–185, 1990.

[16] Cruz F.B., G.R. Mateus, J.M. Smith. A Branch-and-Bound Algorithm to Solve a Multi-level Network Optimization Problem. *Journal of Mathematical Modelling and Algorithms*, 2(1):37–56, 2003.

[17] Dash Associates. Xpress-Mosel Reference Manuals and Xpress-Optimizer Reference Manual, Release 2004G, 2004.

[18] Eckstein J., P.L. Hammer, Y. Liu, M. Nediak, B. Simeone. The Maximum Box Problem and its Application to Data Analysis. *Computational Optimization and Applications*, 23(3):285–298, 2002.

[19] Efroymson M.A., T.L. Ray. A branch and bound algorithm for plant location. *Operations Research*, 14:361–368, 1996.

[20] Fischetti M., A. Lodi, P. Toth. Solving Real-World ATSP Instances by Branch-and-Cut. *Lecture Notes In Computer Science*, pages 64–77, 2003.

[21] Günlük O. A branch-and-cut algorithm for capacitated network design problems. *Mathematical Programming*, 86(1):17–39, 1999.

[22] Hammer P.L., I. Rosenberg, S. Rudeanu. On the determination of the minima of pseudo-boolean functions. *Studii si Cercetari Matematice*, 14:359–364, 1963.

[23] Hammer P.L., R. Holzman. Approximations of Pseudo-Boolean Functions: Applications to Game Theory. *ZOR – Methods and Models of Operations Research*, 36:3–21, 1992.

[24] Hammer P.L., S. Rudeanu. Pseudo-Boolean Programming. *Operations Research*, 17(2):233–261, 1969.

[25] Hansen E.R. *Global optimization using interval analysis.* Marcel Dekker New York, 2004.

[26] Hoffman K.L., M. Padberg. Solving Airline Crew Scheduling Problems by Branch-and-Cut. *Management Science*, 39(6):657–682, 1993.

[27] Holmberg K., D. Yuan. A Lagrangean Heuristic Based Branch-and-bound Approach for the Capacitated Network Design Problem. *Operations Research*, 48(3):461–481, 2000.

[28] Ibaraki T. Theoretical comparisons of search strategies in branch-and-bound algorithms. *International Journal of Parallel Programming*, 5(4):315–344, 1976.

[29] Land A.H., A.G. Doig. An Automatic Method of Solving Discrete Programming Problems. *Econometrica*, 28(3):497–520, 1960.

[30] Lavrac N. Subgroup Discovery Techniques and Applications. In *Advances in Knowledge Discovery and Data Mining*, volume 3518 of *Lecture Notes in Computer Science*, pages 2–14, 2005.

[31] Li C.M., F. Manya, J. Planes. Exploiting Unit Propagation to Compute Lower Bounds in Branch and Bound Max-SAT Solvers. *Proc. of the 11 thCP, Sitges, Spain*, 2005.

[32] Nauss R.M. An Improved Algorithm for the Capacitated Facility Location Problem. *The Journal of the Operational Research Society*, 29(12):1195–1201, 1978.

[33] Nemhauser G.L., Wolsey L.A. *Integer and combinatorial optimization*. Wiley-Interscience New York, NY, USA, 1988.

[34] Ortega F., L.A. Wolsey. A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem. *Networks*, 41(3):143–158, 2003.

[35] Padberg M., G. Rinaldi. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. *SIAM Review*, 33(1):60–100, 1991.

[36] Padberg M., G. Rinaldi. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. *SIAM Review*, 33(1):60–100, 1991.

[37] Palacios H., H. Geffner. Planning as branch and bound: A constraint programming implementation. *Proceedings of CLEI*, 2, 2002.

[38] Pascheuer N., M. Jünger, G. Reinelt. A Branch & Cut Algorithm for the Asymmetric Traveling Salesman Problem with Precedence Constraints. *Computational Optimization and Applications*, 17(1):61–84, 2000.

[39] Gomory R.E. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[40] Gomory R.E. An algorithm for the mixed integer problem. *Report RM-2597, The Rand Corporation*, 1960.

[41] Savelsbergh M. A Branch-And-Price Algorithm for the Generalized Assignment Problem. *Operations Research*, 45(6):831–841, 1997.

[42] Savelsbergh M.W.P. *Preprocessing and Probing Techniques for Mixed Integer Programming Problems*. Eindhoven University of Technology, Department of Mathematics and Computing Science, 1992.

[43] Senne E.L.F., L.A.N. Lorena, M.A. Pereira. A branch-and-price approach to p-median location problems. *Computers and Operations Research*, 32(6):1655–1664, 2005.

[44] Tawarmalani M., N.V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99(3):563–591, 2004.

[45] Vance P.H., C. Barnhart, E.L. Johnson, G.L. Nemhauser. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications*, 3(2):111–130, 1994.

[46] Vandenbussche D., G.L. Nemhauser. A branch-and-cut algorithm for nonconvex quadratic programs with box constraints. *Mathematical Programming*, 102(3):559–575, 2005.

[47] Vidal V., H. Geffner. Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. *Artificial Intelligence*, 170(3):298–335, 2006.

	Obj. Function Value		Running Time (s)	
Instance	Xpress-MP	BBPBF	Xpress-MP	BBPBF
p_20_1	3.86*	3.86*	0.4	0.0
p_20_2	3.69*	3.69*	0.3	0.0
p_20_3	4.44*	4.44*	0.4	0.1
p_30_1	5.11*	5.11*	2.0	0.4
p_30_2	6.37*	6.37*	1.6	0.4
p_30_3	9.19*	9.19*	1.3	0.1
p_40_1	6.62	7.50*	4.1	1.0
p_40_2	7.25	7.25*	4.1	1.3
p_40_3	5.11*	5.11*	3.8	1.6
p_50_1	7.92	8.27	6.3	1.1
p_50_2	5.85	6.61	5.8	3.0
p_50_3	6.45	7.46	5.5	2.5
p_70_1	5.69	8.74	9.1	1.3
p_70_2	8.29	8.86	9.6	1.3
p_70_3	9.85	11.37	10.5	2.6
p_100_1	7.33	8.22	14.0	1.7
p_100_2	9.86	9.90	15.1	1.8
p_100_3	11.87	20.40*	14.2	33.1
p_150_1	8.30	11.66	27.9	4.1
p_150_2	10.54	12.71	26.9	4.3
p_150_3	8.96	10.75	29.7	3.8
p_200_1	11.10	12.09	48.6	7.6
p_200_2	10.57	10.97	49.7	7.9
p_200_3	6.70	9.95	45.4	7.5
p_250_1	8.39	14.53	82.6	14.0
p_250_2	7.19	16.56	81.5	14.6
p_250_3	6.91	16.79	85.1	15.2
p_300_1	9.90	17.90	220.2	29.0
p_300_2	7.60	14.49	229.7	27.0
p_300_3	5.43	18.94	229.9	32.7

Table 2: Quality of solutions for randomly generated instances of problem (PB) using a maximum of 1,000 branch-and-bound nodes. Objective function values marked with an asterisk (*) were proven to be optimal during the truncated search.

	Obj. Function Value		Running Time (s)	
Instance	Xpress-MP	BBPBF	Xpress-MP	BBPBF
p_40_1	7.50*	—	6.3	—
p_40_2	7.25*	—	5.9	—
p_40_3	—	—	—	—
p_50_1	7.92	8.27	9.2	4.3
p_50_2	5.88	6.61*	10.3	4.5
p_50_3	6.98	7.46*	10.6	3.3
p_70_1	6.79	8.75	14.8	5.4
p_70_2	8.29	9.03	13.7	6.0
p_70_3	10.15	11.37	16.0	9.2
p_100_1	7.33	8.39	23.1	3.3
p_100_2	10.49	9.90	22.3	3.4
p_100_3	14.66	—	22.6	—
p_150_1	8.30	13.38	46.3	40.1
p_150_2	10.54	12.71	46.8	8.1
p_150_3	9.35	12.65	46.7	16.0
p_200_1	14.06	14.70	84.2	14.5
p_200_2	10.87	10.97	82.4	11.0
p_200_3	10.77	13.11	80.5	11.4
p_250_1	8.39	14.53	132.1	18.2
p_250_2	8.17	16.56	157.0	70.0
p_250_3	11.48	18.31	155.8	39.8
p_300_1	9.90	20.32	356.6	250.0
p_300_2	7.60	14.49	365.4	59.2
p_300_3	5.43	18.94	350.4	92.0

Table 3: Quality of solutions for randomly generated instances of problem (PB) using a maximum of 2,000 branch-and-bound nodes. The problems with at most 30 variables are not shown here because each of them was solved to optimality by both algorithms using at most 1,000 branch-and-bound nodes. Objective function values marked with an asterisk (*) were proven to be optimal during the truncated search. Instances marked with "—" were solved to optimality using at most 1,000 branch-and-bound nodes.

	After 1,000 nodes		After 2,000 nodes	
Instance	Xpress-MP	BBPBF	Xpress-MP	BBPBF
p_20_1	3.86*	3.86*	—	—
p_20_2	3.69*	3.69*	—	—
p_20_3	4.44*	4.44*	—	—
p_30_1	5.11*	5.11*	—	—
p_30_2	6.37*	6.37*	—	—
p_30_3	9.19*	9.19*	—	—
p_40_1	9.53	7.50*	7.50*	—
p_40_2	9.70	7.25*	7.25*	—
p_40_3	5.11*	5.11*	—	—
p_50_1	16.35	16.96	15.29	14.51
p_50_2	10.61	11.03	9.34	6.61*
p_50_3	11.38	13.54	10.23	7.46*
p_70_1	18.90	18.73	17.85	15.57
p_70_2	17.61	18.05	16.58	15.87
p_70_3	22.79	21.79	21.72	18.38
p_100_1	26.18	24.63	25.47	22.70
p_100_2	26.36	25.32	25.74	24.74
p_100_3	33.39	20.40*	32.60	—
p_150_1	34.70	31.11	34.22	30.73
p_150_2	33.79	30.75	33.10	29.51
p_150_3	36.92	31.90	36.22	31.42
p_200_1	51.55	44.85	50.92	44.20
p_200_2	43.85	36.62	43.19	36.59
p_200_3	47.96	40.46	47.00	40.08
p_250_1	55.12	45.80	54.69	45.48
p_250_2	56.17	47.10	55.72	45.90
p_250_3	59.37	50.39	58.89	50.08
p_300_1	75.61	61.76	75.13	60.38
p_300_2	73.35	62.17	72.89	60.46
p_300_3	71.92	59.97	71.22	57.92

Table 4: Upper bound on the value of the optimal solution of randomly generated instances of problem (PB). Instances marked with an asterisk were solved to optimality.

	Objective Function Value		Running Time (s)	
Instance	Xpress-MP	BBPBF	Xpress-MP	BBPBF
p_40_1	7.50*	7.50*	6.0	1.0
p_40_2	7.25*	7.25*	5.8	1.3
p_40_3	5.11*	5.11*	3.7	1.6
p_50_1	8.27*	8.27*	55.9	9.0
p_50_2	6.61*	6.61*	19.8	4.5
p_50_3	7.46*	7.46*	19.2	3.3
p_70_1	8.75*	8.75*	237.3	26.6
p_70_2	9.03*	9.03*	189.7	23.8
p_70_3	11.37*	11.37*	190.3	18.2
p_100_1	11.00*	11.00*	1,281.4	317.9
p_100_2	11.11*	11.11*	1,203.1	236.2
p_100_3	20.40*	20.40*	572.4	33.2
p_150_1	11.88	14.66*	1,825.3	1,477.8
p_150_2	12.78	12.71	1,826.4	1,800.6
p_150_3	11.61	13.65	1,823.3	1,801.7
p_200_1	18.97	20.54	1,823.2	1,801.3
p_200_2	14.67	17.62	1,827.1	1,800.5
p_200_3	15.83	14.42	1,824.9	1,800.5
p_250_1	14.22	18.60	1,820.8	1,800.2
p_250_2	14.53	18.83	1,826.5	1,801.0
p_250_3	15.16	19.67	1,825.9	1,800.5
p_300_1	13.62	20.32	1,820.8	1,800.1
p_300_2	12.65	19.93	1,824.5	1,800.3
p_300_3	11.73	18.94	1,832.5	1,800.1

Table 5: Results of the truncated search with a limit of 30 minutes on the running time and no limit on the number of branch-and-bound nodes. Instances marked with an asterisk were solved to optimality.

	Number of nodes		Upper Bound	
Instance	Xpress-MP	BBPBF	Xpress-MP	BBPBF
p_40_1	1,668	369	7.50	7.50
p_40_2	1,570	603	7.25	7.25
p_40_3	942	885	5.11	5.11
p_50_1	12,864	3,098	8.27	8.27
p_50_2	3,995	1,237	6.61	6.61
p_50_3	3,825	1,242	7.46	7.46
p_70_1	32,587	4,574	8.75	8.75
p_70_2	22,031	4,112	9.03	9.03
p_70_3	35,049	2,633	11.37	11.37
p_100_1	145,547	22,904	11.00	11.00
p_100_2	139,567	15,557	11.11	11.11
p_100_3	71,388	970	20.40	20.40
p_150_1	137,138	22,270	25.87	14.66
p_150_2	121,295	30,059	24.78	23.72
p_150_3	138,396	28,812	27.91	26.69
p_200_1	79,278	12,905	44.33	39.66
p_200_2	77,706	11,547	36.43	34.02
p_200_3	79,233	16,591	40.42	37.05
p_250_1	50,739	10,439	50.00	42.97
p_250_2	53,361	7,477	50.15	44.63
p_250_3	44,891	8,358	54.18	46.45
p_300_1	20,118	5,635	71.94	59.38
p_300_2	18,470	7,426	70.12	58.03
p_300_3	19,004	7,028	68.07	56.95

Table 6: Results of the truncated search with a limit of 30 minutes on the running time and no limit on the number of branch-and-bound nodes.