Duality, Warm Starting, and Sensitivity Analysis in Integer Linear Programming

T. K. Ralphs
M. Guzelsoy
Lehigh University

Report No. 04T-022

Duality, Warm Starting, and Sensitivity Analysis in Integer Linear Programming

T.K. Ralphs* M. Guzelsoy †

November 15, 2004

Abstract

We discuss procedures for sensitivity analysis and warm starting that can be integrated with modern solvers for mixed-integer linear programming problems. The implementation of the methods within the SYMPHONY solver for mixed-integer linear programs is discussed and computational results are presented.

1 Introduction

Duality has long been a central theme in optimization theory. The study of duality has lead to efficient procedures for computing bounds, is central to our ability to perform post facto solution analysis, is the basis for procedures such as column generation and reduced cost fixing, and has provided us with a wide range of useful optimality conditions. Optimality conditions, in turn, can be used to construct "warm starting" procedures that accelerate solution of a problem instance by taking advantage of information obtained during solution of a related instance. Such procedures are useful both in cases where the input data are subject to fluctuation after the solution procedure has been initialized and in cases where the solution of a series of closely related instances is required. A variety of integer optimization algorithms consist of solving a series related mixed-integer linear programs (MILPs). This is the approach, for example, taken by decomposition algorithms, parametric and stochastic programming algorithms, multi-criteria optimization algorithms and algorithms for analyzing infeasible mathematical models.

The study of these topics is thus important to the advancement of the theory and practice of optimization. However, relatively little is known about them with respect to discrete optimization problems. In this abstract, we extend some of the early work in this area, and discuss the integration of the resulting methodology with modern solvers. Following the paradigm provided by the theory of linear programming (LP), our approach is to derive an optimal solution to a particular (strong) dual problem as a by-product of the branch and bound procedure, as suggested by Wolsey [18]. As in LP, this dual solution provides a proof of optimality, can be used to determine the effect on the optimal value when the problem data is perturbed, and can be used as the basis for a warm starting procedure.

*Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18015, tkralphs@lehigh.edu, http://coral.ie.lehigh.edu/~ted

†Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18015, magh@lehigh.edu, http://coral.ie.lehigh.edu/~menal

2 Integer Programming Duality

2.1 Duals for Integer Programs

We first focus on the theory of duality for integer programming, as this is the basis for all of the techniques introduced herein. Although the development of duality theory and sensitivity analysis for MILPs received a good deal of attention in the 1970s and early 1980s, this research area has seen few papers over the past two decades. In light of recent progress in computational methods for MILP, the continued development of these ideas is merited. We briefly introduce some notation and state assumptions. For brevity, we will not define standard terminology, but refer the reader to [5] for definitions. A linear programming problem (LP) is that of minimizing a linear objective function represented by $c \in \mathbb{Q}^n$ over the polyhedral feasible region

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}, \tag{1}$$

defined by constraint matrix $A \in \mathbb{Q}^{m \times n}$ and right-hand side vector $b \in \mathbb{Q}^m$. A MILP is an LP in which a specified subset of the variables are constrained to take on integer values. Without loss of generality, we assume that the variables indexed 1 through $p \leq n$ are the integer variables, so that the feasible region is $\mathcal{P}^I = \mathcal{P} \cap \mathbb{Z}^p \times \mathbb{R}^{n-p}$. For simplicity, we assume throughout that the feasible region is bounded and nonempty, although this assumption is easily removed.

The mixed-integer linear programming problem (the *primal problem*) is to compute the optimal value

$$z_{IP} = \min_{x \in \mathcal{P}^I} c^\top x. \tag{2}$$

Generally, any optimization problem

$$z_D = \max\{g(u) \mid u \in U\} \tag{3}$$

such that $z_D \leq z_{IP}$ is called a *dual problem* and is a *strong dual* if $z_D = z_{IP}$. Obviously, any feasible solution to a dual problem provides a valid lower bound on the optimal value of a solution to the primal problem. A more concrete notion of duality is obtained by considering a dual problem in which one tries to construct a function that return a lower bound on the optimal value, as an explicit function of the input data. This approach results in the rather general dual suggested by Wolsey [18]:

$$z_D^g = \max_{g:\mathbb{R}^m \to \mathbb{R}} \{g(b) \mid g(Ax) \leq c^\top x, x \in \mathcal{P}^I\} = \max_{g:\mathbb{R}^m \to \mathbb{R}} \{g(b) \mid g(d) \leq z_{IP}(d), d \in \mathbb{R}^m\}. \tag{4}$$

Here, $z_{IP}(d) = \min_{x \in \mathcal{P}^I(d)} c^\top x$ is the *value function*, which expresses the optimal value of a MILP as a function of the right-hand side d and $\mathcal{P}^I(d) = \{x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \mid Ax = d, x \geq 0\}$. Solutions to the above program are functions that approximate the value function from below. Optimal solutions are those that agree with the value function at b. It is clear that not all optimal solutions bound the value function equally well. In fact, there are optimal solutions to this dual that provide arbitrarily poor estimates of the value function, even in a local neighborhood of b.

The dual (4) is strong because setting $g^*(d) = z_{IP}(d)$ when $\mathcal{P}^I(d) \neq \emptyset$ and $g^*(d) = 0$ otherwise yields an optimal solution. Blair and Jeroslow [2] derived a closed form for the value function of a pure integer program, essentially by functionally encoding a Gomory cutting plane proof of

optimality to obtain a so-called *Gomory formula*. A Gomory formula may be composed of an exponential number of nested floor functions. Computing such a function for a specific right-hand side is equivalent to solving the original integer program by Gomory's cutting plane algorithm. This result can be extended to MILPs by applying Bender's decomposition to arrive at a more complex class of functions known as *Jeroslow formulas*. The value function is ideal from the perspective of obtaining good lower bounds, but it does not arise in a natural way as a by-product of branch and bound, the algorithm most commonly use for solution of MILPs.

It is natural to consider whether it is possible to restrict the class of functions considered in (4) in some reasonable way. If we restrict g to be linear, then an optimal solution to (4) is $g^*(d) = \max\{u^\top d \mid u^\top A \leq c\}$, which is the dual of the continuous relaxation of the original MILP. Hence, this restriction results in a dual that is no longer strong. Jeroslow showed that restricting to the class of convex functions still results in the dual above [6]. In a series of papers, Johnson [9, 7, 8] and later Jeroslow [6] developed the idea of restricting the domain to the set of subadditive functions[1]. The subadditive functions are a superset of the linear functions that retain the intuitively pleasing property of "no increasing returns to scale" associated with linear functions. With this restriction, we can rewrite (4) in the pure integer case as the *subadditive dual*

$$z_D^s = \max\{F(b) \mid F(a^i) \leq c_i, F \text{ subadditive}\}, \tag{5}$$

where a^i is the i^{th} column of the matrix A. Despite the restriction, this dual is still strong since the value function is subadditive if we restrict the domain to all right-hand sides for which the original MILP is feasible. Blair showed that the value function can be extended to a subadditive function defined on all of $\mathbb{R}^m$ [1]. In [18], Wolsey showed that the subadditive dual extends many of the properties of the LP dual, such as complementary slackness and the concept of "reduced cost," to MILP. Unfortunately, there is no general method for efficiently deriving optimal solutions in the pure integer case and little is known about the mixed-integer case.

2.2 Dual Solutions from Branch and Bound

We now consider methods for producing dual information as a by-product of the branch and bound algorithm. We restrict ourselves here to consideration of *simple branch and bound*, in which the partitioning is done only by adjusting the bounds on variables, so that bound changes arising from branching can be handled implicitly. We also assume that no preprocessing or other logical procedure, such as reduced cost fixing or cutting plane generation, is used to tighten the LP relaxation.

For linear programs solved using the simplex algorithm, dual solutions can be obtained by constructing an *optimal basis*, which is a nonsingular submatrix B of A for which $B^{-1}b \geq 0$ (primal feasibility) and $c - c_B B^{-1} A \geq 0$ (dual feasibility), where c_B are the components of c corresponding to the columns of B. For MILPs, this notion can be extended to that of an *optimal partition*, an idea explored by Skorin-Kapov and Granot for quadratic programs in [4]. Consider a partition of $\mathcal{P}$ into the subpolyhedra $\mathcal{P}_1, \ldots, \mathcal{P}_s$ in such a way that $\mathcal{P}^I \subseteq \cup_{i=1}^s \mathcal{P}_i$ and assume that these subpolyhedra are nonempty. Let LP_i be the linear program $\min_{x^i \in \mathcal{P}_i} c^\top x^i$ associated with the subpolyhedron $\mathcal{P}_i$. The usual optimality conditions for branch and bound are captured formally in the following observation.

[1] A function $f : \mathbb{R}^m \to \mathbb{R}$ is subadditive over domain D if $f(x+y) \leq f(x) + f(y)$ for all $x, y \in D$.

Observation 1 *Let B^i be an optimal basis for LP_i. Let*

$$U = \min\{c_{B^i}(B^i)^{-1}b + \beta_i \mid 1 \leq i \leq s, \hat{x}^i \in \mathcal{P}^I\} \tag{6}$$

and

$$L = \min\{c_{B^i}(B^i)^{-1}b + \gamma_i \mid 1 \leq i \leq s\}, \tag{7}$$

where β_i and γ_i are the constant factors associated with the nonbasic variables fixed at nonzero bounds and $\hat{x}^i$ is the BFS corresponding to basis B^i. If $U = L$, then $z_{IP} = U$ and for each $1 \leq j \leq s$ such that $\hat{x}^j \in \mathcal{P}^I$ and $c_{B^j}(B^j)^{-1}b = z_{IP}$, $\hat{x}^j$ is an optimal solution.

The goal of the branch and bound algorithm is to produce such a partition through a recursive partitioning scheme. The product of the algorithm is a tree whose leaves represent members of a partition and whose internal nodes represent subpolyhedra that were subsequently further subdivided.

Expressing the quantity L above as a function of d, the right-hand side vector yields the following optimal solution to (4) when it is made real-valued as before:

$$L(d) = \min\{c_{B^i}(B^i)^{-1}d + \gamma_i \mid 1 \leq i \leq s\}. \tag{8}$$

Unfortunately, this function is not subadditive and is therefore not a feasible solution to (5). However, it can still be used quite effectively for sensitivity analysis and warm starting. Most importantly, it can be readily computed from data produced as a by-product of the branch and bound algorithm and provides a lower bound on the value of an optimal solution for any right-hand side vector d. A similar function of the objective function vector c can be used to yield a valid upper bound after changes to that vector. The function (8) can also be extended to account for empty subpolyhedra.

3 Sensitivity Analysis

Sensitivity analysis procedures follow naturally from the duality theory just presented. Such procedures can be broken down into two broad categories, *local* and *global.*

3.1 Local Sensitivity Analysis

Local sensitivity analysis is concerned with the effect of relatively small perturbations to the original data. For certain classes of perturbation, it is easy to show that optimality is retained. For example, increasing the objective function value of a variable already at its lower bound will certainly not effect optimality of the current solution. A number of such rules are derived for integer programming by Geoffrion and Nauss [3]. For other types of perturbations, the most straightforward approach is to consider the effect on optimality conditions of perturbations to the input data.

Changes to the Right-hand Side. For simple branch and bound, one can examine each member of the optimal partition to determine ranges over which the right-hand side coefficients can be changed without affecting the optimality conditions, as in linear programming. Changes to the right-hand side do not affect the dual feasibility of each basis, so the relevant range is that over which primal feasibility is maintained. After determining a valid range for each member of the

partition, one must then take the intersection of these ranges to determine an overall valid range. Since dual feasibility of each basis in the partition is maintained, one can derive lower bounds outside of the valid range by applying the dual function (8). For a particular right-hand side, one can improve on this bound by actually calculating new optimal bases, or continuing the partitioning process (see Section 4). In the case of a change outside the valid range, one must of course account for situations in which previously nonempty members of the partition become empty. This can be dealt with in a straightforward manner by considering the result of solving a Phase I LP, in the case of the primal simplex algorithm, or the proof of infeasibility returned by the dual simplex algorithm.

Changes to the Objective Function. For simple branch and bound, changing the objective function vector leaves the bases in the partition primal feasible, but may invalidate dual feasibility. This means that the upper bound obtained from the optimal partition is still valid, while the lower bound may not be. Again, valid ranges can be calculated by considering a range for each basis in the partition and then taking their intersection. Outside the valid range, a lower bound may still be obtained by computing a new optimal basis for each member of the partition. Since the feasible region is unchanged and an initial feasible basis is available, this calculation can be performed efficiently. One may note that the upper bound obtained here may be relatively weak, since generally few feasible solutions are obtained during the solution process. Even if multiple solutions are obtained, most solvers return only one. To address this problem, one may want to explicitly track all solutions found during the search for later use during sensitivity analysis. Taking this idea a step further, Geoffrion and Nauss also suggest modifying the branch and bound algorithm to explicitly return not just the optimal solution, but the top k solutions [3]. This can be done in a fairly straightforward manner.

Other Changes. Adding a constraint is similar to modifying the right-hand side, except that one must expand the current basis by adding the slack or artificial variable for the new constraint to the basis for each member of the partition. Deleting a constraint is a bit more involved. The basis for each member of the partition must be reduced in this case. The easiest way to achieve this is to first rotate the slack or artificial variable into the basis. This variable can then be deleted, along with the constraint. Adding a variable is similar to modifying the objective function. The variable is simply added to each member of the partition. The basis remains primal feasible, but may no longer be dual feasible. To delete a variable, the variable must be rotated out of the basis in each partition and then removed. General modifications to the constraint matrix are more difficult because the basis may not remain valid, i.e., may no longer be nonsingular. We have been able to show that the basis remains valid under certain structured changes to the constraint matrix that occur in multi-criteria optimization, but in many cases, it is necessary to construct a new optimal basis from scratch for each member of the partition.

3.2 Global Sensitivity Analysis

In global sensitivity analysis, one considers how the optimal value of a MILP changes globally as a function of either the objective vector or the right-hand side vector (or possibly both). For LPs, the global sensitivity functions are continuous and piecewise linear and can be written in closed

form. For MILPs, the function that expresses the optimal value as a function of the objective vector remains piecewise linear and continuous, but the value function discussed earlier, although it has a closed form, is more complex. Nonetheless, there are algorithmic approaches to accessing information about the global sensitivity functions without actually constructing them by using parametric programming.

A *parametric MILP* is defined to be a family of MILPs parameterized by a single scalar θ. For instance,

$$z_{IP}(\theta) = \min_{x \in \mathcal{P}^I(\theta)} c^\top x \tag{9}$$

where $\mathcal{P}^I(\theta)$ is a parameterized family of polyhedra, is an example of a parametric MILP. One can also parameterize the objective function or even the constraint matrix. The goal is to determine the complete set of optimal values that occur as θ is varied. This set characterizes a one-dimensional "slice" of the global sensitivity function.

A large number of papers have been written on how to analyze parametric programs, especially those in which the parameterization is on the right-hand side vector. Our approach to analyzing such programs is to formulate a corresponding bicriteria optimization problem. As an example, suppose we wish to analyze the family of MILPs with the parametric objective $c + \theta d$ for $d \in \mathbb{R}^n$. To do so, we solve a bicriteria MILP where the two objective functions considered are c and d. We can analyze MILPs with parameterized right-hand sides in a similar fashion. In [12], we presented an asymptotically optimal algorithm for determining the set of supported solutions for a bicriteria MILP. Computational results obtained using this algorithm are presented in Section 6. Methods for analyzing all family members within a single branch and bound tree have also been suggested by Rountree and Gillett [16] and Marsten and Morin [10].

Parametric analyses such as these are limited to a one-dimensional slice of one of the global sensitivity functions. Multi-dimensional slices could be analyzed by extension of these methods to multiple criteria, although analysis in dimensions above two is likely to be prohibitive without a specialized method taking advantage of warm starting. Several authors have reported theoretical results indicating that it may be possible to analyze global, simultaneous changes to the right-hand side and the objective function vector. For instance, Sturmfels and Thomas [17] showed that for a pure integer program, there is a finite collection of objective functions that need to be considered in order to yield all possible sets of optimal solutions as both the right-hand side and objective functions are varied. This result, however, does not have useful algorithmic content.

4 Warm Starting

Methods for warm starting are useful in cases where either (1) we need to solve a family of related MILPs of which we have a priori knowledge, or (2) the input data are uncertain and may change as the solution procedure progresses. Both of these situations arise frequently in practice. In keeping with our general approach, methods for warm starting can be thought of as being based on a specific set of optimality conditions. If we view an optimization algorithm as an iterative scheme for achieving such conditions, then progress of the algorithm can be measured roughly as "distance from optimality," i.e., the degree of violation of the optimality conditions. From this point of view, a *warm start* can be thought as additional input data that allows the algorithm to make fast initial progress by starting closer to optimality.

Our warm start procedure follows naturally from our approach to sensitivity and is similar to a scheme suggested by Roodman for Balas' additive algorithm [15]. For MILPs, the most common measure of distance from optimality is the percentage difference between the upper and lower bounds. As additional input to initialize the algorithm, we provide a branch and bound tree calculated while solving another (related) MILP, and a list of the differences between the model used to generate the tree and the current model to be solved. We will refer to the input tree as a *warm start tree.* The tree provides an initial partition, whose corresponding upper and lower bounds are generally expected to be much better than those that would be obtained from a cold start.

To initialize the algorithm from a warm start tree, it is necessary to obtain an optimal basis for each member of the partition, as described earlier in Section 3. After this, calculations can continue as they normally would in branch and bound by adding the leaf nodes to the set of candidate subproblems awaiting processing. Whenever possible, of course, we utilize the previous optimal basis for each member of the partition to aid in obtaining the new ones. Note that it is not necessary that the initial partition be the one yielded by the leaves of the previous branch and bound tree. In fact, internal nodes can also be considered as potential members of the initial partition. When an internal node is utilized, all descendants of that node are discarded. This may be desirable in cases where the branch and bound tree is large and the partition is too fine to be efficient. In Section 6, we will discuss the determination of a proper warm start tree.

In addition to a warm start tree, we may also provide additional data in some cases. For instance, we may also provide a previously generated global cut pool or a pool of previously generated solutions useful in establishing a priori upper bounds. In some cases, the best strategy may be to use only the cut pool in warm starting and throwing away the warm start tree.

5 Implementation

All of the methods described herein have been implemented within the SYMPHONY 5.0 MILP solver, which is part of the COIN-OR software suite and can be accessed through the COIN-OR Open Solver Interface (OSI). The OSI provides a uniform API to solvers for LPs and MILPs and already supports the concept of warm starting and has an associated base class for storing and loading warm starting information. Using this base class, we have defined a warm start class for MILPs and have implemented the methods needed for SYMPHONY to utilize it. Our current implementation stores the tree in the native format utilized by SYMPHONY, which is a compact representation achieved by storing, for each node, only the differences in description between the node and its parent. This results in an extremely efficient data structure. Our eventual goal is to develop open standards by which warm start information could be shared between solvers, as it generally can be with linear programs. SYMPHONY can be stopped at any time and a warm start tree saved, then reloaded for later use. Only pure branch and bound is fully supported, but we are working on extensions, as described in Section 7. Details of the use and implementation of these methods can be found in the SYMPHONY 5.0 User's Manual [14] and in a recent proceedings paper [11].

6 Computational Results

We focus here on preliminary computational results with the warm starting procedures we have described, as implemented in the publicly available SYMPHONY 5.0. The computational platform for all tests was a 4-processor SMP machine with Intel Xeon 2.4GHz processors and 2G of memory. All tests were run with the sequential version of SYMPHONY. We tested the warm starting capabilities for two different applications. In the first set of experiments, we tested the use of warm starting while using SYMPHONY's bicriteria solver to analyze the tradeoff between fixed and variable costs for a class of network routing problems described in [12] and [13]. Because the algorithm involves solving a sequence of related single-criteria MILPs, it is natural to consider the use of warm starting to accelerate solution of the subproblems. The question that arises immediately, however, is exactly how to generate the warm start information. In contrast to the linear programming case, we have a wide variety of choices for a starting partition and it is not clear a priori what makes a "good" starting partition. There are two main questions to be answered. First, from what previous calculation should we take the starting partition? In the case of multi-criteria optimization, we have the choice of deriving a starting partition from the branch and bound trees produced during solution of any previous subproblem. Second, given a particular previously generated branch and bound tree, what portion of the tree should be used to derive the starting partition? Again, we can choose any subtree of the given tree, as described earlier.

For multi-criteria problems, the single objective used in each subproblem is a convex combination of the two original objectives. Therefore, in answer to the first question above, we derive our starting partition from the previously solved subproblem for which the weight determining the combination is closest to that of the current subproblem. In answer to the second question, we choose the warm start tree to be the first α percent of the nodes generated during the procedure that generated the original tree. Because a starting partition that is too fine may actually be detrimental, we tried values of α ranging from 0 to 100. A value of 20 seemed to be the best and resulted in a uniform improvement in running times. The results reported in Table 1 are for the instances of size 15 reported in [13]. Note that we did allow cut generation for these runs, but this is not an issue, as long as the validity of the cuts is not affected by the change in objective function. In Table 1, the first chart compares α values of 0 and 20, whereas the second one compares α values of 0 and 100. The results with $\alpha = 20$ show a clear and marked improvement over those with $\alpha = 0$. Results with $\alpha = 100$ show a detrimental effect in some cases, but an overall improvement.

In a second set of experiments, we tested the use of warm starting to solve selected instances from MIPLIB 3 after randomly perturbing the objective function vector by changing each coefficient by a random percentage between $-\alpha$ and α. Since the warm starting procedure is likely to have a smaller effect for large perturbations, we report in Table 2 on experiments with values of α equal to 1, 10, and 20. It is evident that the results are better for smaller values of α, but it is also clear that warm starting has an overall positive effect with some exceptions for larger values of α. Further experiments with warm starting in stochastic integer programming are reported in [11].

7 Conclusions and Future Work

We have outlined a theory and methodology for performing sensitivity analysis and warm starting computations that can be integrated with the branch and bound algorithm used by modern

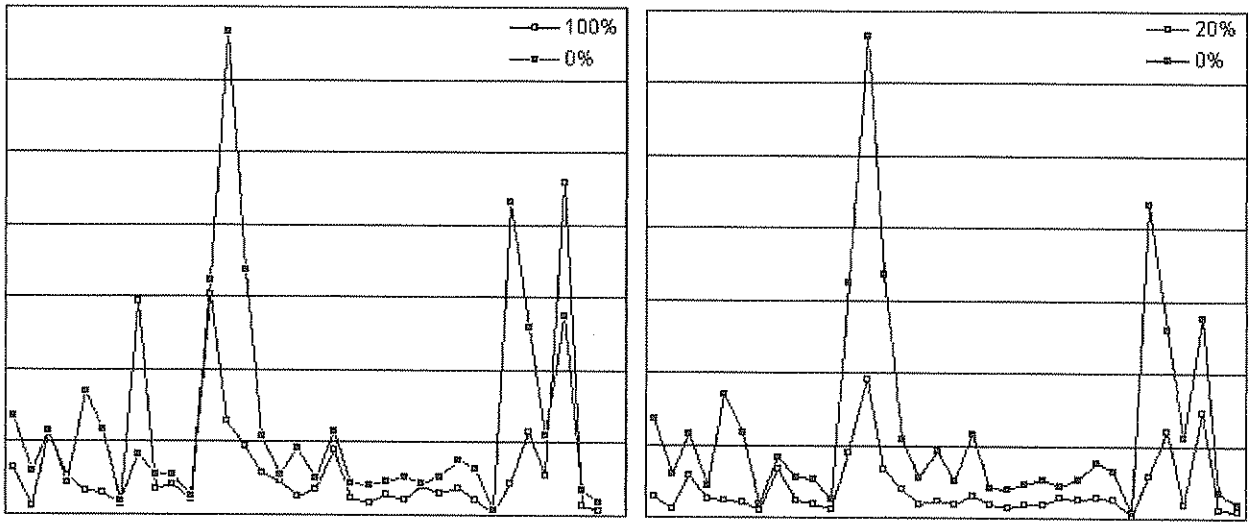

Table 1: Results of using warm starting to solve multi-criteria optimization problems.

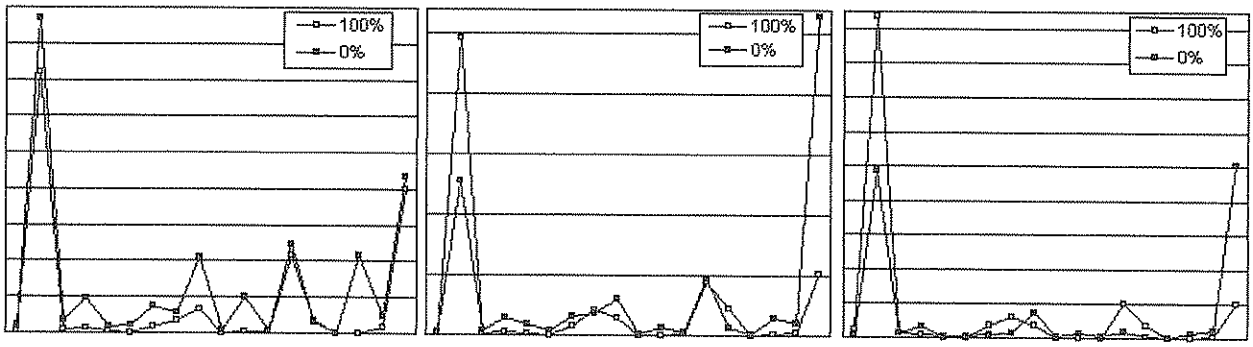

Table 2: Results of using warm starting to re-solve MILPs with $\alpha = 1, 10, 20$.

solvers for mixed-integer linear programs. This work is intended to lay the foundation for more significant advances, both theoretical and computational, to follow. In principle, these techniques can be extended to the more sophisticated variants of branch and bound employed by most modern solvers, but the implementation of such techniques is challenging and requires a more sophisticated approach. Even for the case of simple branch and bound, many questions regarding the most efficient ways to employ these techniques in practice remain to be answered. We have only scratched the surface so far, but our preliminary results indicate that there are important potential gains to be made by developing these ideas further.

References

[1] BLAIR, C. Extensions of subadditive functions used in cutting plane theory. MSSR No. 360, Carnegie Melln University, 1974.

[2] BLAIR, C., AND JEROSLOW, R. The value function of an integer program. *Mathematical Programming 23* (1982), 237–273.

[3] GEOFFRION, A., AND NAUSS, R. Parametric and postoptimality analysis in integer linear programming. *Management Science 23* (1977), 453–466.

[4] GRANOT, F., AND SKORIN-KAPOV, J. Some proximity and sensitivity results in quadratic integer programming. *Mathematical Programming 47* (1990), 259–268.

[5] GREENBERG, H. Mathematical programming glossary. http://carbon.cudenver.edu/ hgreenbe/glossary/index.php.

[6] JERSOLOW, J. Cutting plane theory: Algebraic methods. *Discrete Mathematics 23* (1978), 121–150.

[7] JOHNSON, E. Cyclic groups, cutting planes, and shortest paths. In *Mathematical Programming*, T. Hu and S. Robinson, Eds. Academic Press, New York, 1973.

[8] JOHNSON, E. On the group problem and a subadditive approach to integer programming. *Annals of Discrete Mathematics 5* (1979), 97–12.

[9] JOHNSON, E. L. On the group problem for mixed integer programming. *Mathematical Programming Study 2* (1974), 137–179.

[10] MARSTEN, R., AND MORIN, T. Parametric integer programming: The right-hand side case. *Annals of Discrete Mathematics 1* (1977), 375–390.

[11] RALPHS, T., AND GUZELSOY, M. The SYMPHONY callable library for mixed-integer linear programming. To appear in the Proceedings of the Ninth INFORMS Computing Society Conference, 2004.

[12] RALPHS, T., SALTZMAN, M., AND WIECEK, M. An improved algorithm for biobjective integer programming. To appear in Annals of Operations Research, 2004.

[13] RALPHS, T., SALTZMAN, M., AND WIECEK, M. An improved algorithm for biobjective integer programming and its application to network routing problems. Technical Report 04T-004, Lehigh University Industrial and Systems Engineering, 2004.

[14] RALPHS, T. K. SYMPHONY Version 5.0 user's manual. Technical Report 04T-011, Lehigh University Industrial and Systems Engineering, 2004.

[15] ROODMAN, G. Postoptimality analysis in zero-one programming by implicit enumeration. *Naval Research Logistics Quarterly 19* (1972), 435–446.

[16] ROUNTREE, S., AND GILLETT, B. Parametric integer linear programming: A synthesis of branch and bound with cutting planes. *European Journal of Operational Research 10* (1982), 183–189.

[17] STURMFELS, B., AND THOMAS, R. Variation of cost functions in integer programming. *Mathematical Programming 77* (1997), 357–387.

[18] WOLSEY, L. Integer programming duality: Price functions and sensitivity analysis. *Mathematical Programming 20* (1981), 173–195.