NSF GRANT #DMI-0534862
NSF PROGRAM NAME: Small Grants for Exploratory Research

Duality and Warm Starting in Integer Programming

Lead P.I. Ted Ralphs
Lehigh University

Menal Güzelsoy
Lehigh University

Abstract: Integer programs are mathematical programming models in which some or all of the variables are required to take on values in a discrete set. Such mathematical models arise in a wide variety of applications, such as supply chain optimization, portfolio optimization, production and logistics problems, medical applications, security applications, and more. Many of the settings in which integer programming is applied in practice are dynamic in nature and the need to solve sequences of very similar integer programs arises frequently. This is both as a result of unpredicted changes in the input data and the possible need to analyze multiple alternative scenarios. For this reason, the development of procedures that can utilize information from the solution of one integer program to accelerate later solution of slightly modified instances is extremely important in practice. The main focus of this research is to develop prototypes for such procedures and to integrate them with a modern solver for mixed-integer linear programming problems. Because warm starting procedures and other related sensitivity analysis procedures are inherently based on duality, we have first undertaken a brief study of integer programming duality. Below, we report both on this study and our experience so far implementing a warm starting methodology within the SYMPHONY callable library, an extensive software framework developed under a previous NSF grant.

1. Introduction: Duality has long been a central theme in optimization theory. The study of duality has led 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 initiated 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 taken, for example, 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 paper, 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, our approach is to construct an optimal solution to a particular (strong) dual problem as a by-product of the branch-and-bound procedure, as suggested by Wolsey [1]. As in linear programming, 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.

2. Integer Programming Duality: We first present a brief survey of and some updates to the theory of duality for integer programs. 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, a duality theory that can be better integrated with current computational practice is needed. We briefly introduce some notation and state assumptions. For brevity, we will not define standard terminology, but refer the reader to [2] for definitions.

A linear programming problem (LP) is that of minimizing a linear objective function represented by $c \in \mathbb{Q}^n$ over a polyhedral feasible region

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

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 LP is given as in (1) and that the variables indexed 1 through $p \leq n$ are the integer variables, so that the feasible region is $\Pi = \mathcal{P} \cap (\mathbb{Z}^p \times \mathbb{R}^{n-p})$. We can then state the mixed-integer linear programming problem defined by the triple $(c, \mathcal{P}, p)$ as

$$P = \min_{x \in \Pi} cx. \quad (2)$$

For the remainder of the paper, we will refer to this canonical instance of MILP as the *primal problem*. For simplicity, we assume throughout that the feasible region is bounded and nonempty, although this assumption is easily removed.

The study of duality has two main goals: (1) to derive valid lower bounds on a specific instance of the primal problem (usually to aid in solution of that problem) and (2) to determine the effect of modifications to the input data on the optimal solution and/or optimal solution value. Generally, an optimization problem

$$D = \max_{u \in U} g(u), \quad (3)$$

with objective function $g : U \to \mathbb{R}$ and feasible region $U \subseteq \mathbb{R}^k$ such that $D \leq P$ is called a *dual problem* and is a *strong dual problem* if $D = P$. For any pair (g, U) that comprises a dual problem for P and any $u \in U$, $g(u)$ is a valid lower bound on P and the dual problem is that of finding the best such bound. The usefulness of such a dual problem is rather limited, however, since it may only provide a bound for the single MILP instance being analyzed and since the pair g and U are not selected according to any obvious criterion for goodness.

There are two natural avenues for improvement of this dual framework. First, one can let the dual problem itself vary and try to choose the "best" among the possible alternatives. This leads to the generalized dual

$$D = \max_{g,U} \max_{u \in U} g(u), \quad (4)$$

where each pair (g, U) considered in the above maximization must comprise a dual problem of the form (3). This generalized dual will yield an improved bound, but still does not provide us with an obvious method for analyzing the effect of perturbations to the instance. This leads up to the second avenue for extending our notion of duality, which is to develop procedures for producing not just a lower bound valid for a single instance, but a *dual function* that can produce valid bounds across a range of possible instances within a neighborhood of a given base instance. Such dual functions are needed to allow us to study the effect of perturbations of the input data and to develop warm starting procedures.

Because the right-hand side can be thought of as describing the level of resources available within the system being optimized, it is natural to first consider how the optimal solution value of an integer program changes as a function of the right-hand side. The *value function* for an integer program is a function that returns the optimal solution value to an integer program as a function of the right-hand side, i.e., it is a function $z : \mathbb{R}^m \to \mathbb{R}$ defined by

$$z(d) = \min_{x \in \Pi(d)} cx, \quad (5)$$

where $\Pi(d) = \{x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \mid Ax = d, x \geq 0\}$ for $d \in \mathbb{R}^m$. By convention, $z(d) = \infty$ if $\Pi(d) = \emptyset$.

The value function tells us how the optimal value changes as a function of available resources, so it is an ideal candidate for a dual function. Blair and Jeroslow [3] derived a closed form for the value function of a pure integer program, the so-called *Gomory formulas*, which can be obtained essentially by functionally encoding a Gomory cutting plane proof of optimality. This result can be extended to MILPs by embedding a correction term obtained from an associated LP subproblem to arrive at a more complex class of functions known as *Jeroslow formulas*. Unfortunately, constructing such functions appears to be as difficult as solving the original integer program by Gomory's cutting plane procedure. The value function also does not arise in a natural way as a byproduct of branch and cut, the algorithm most commonly use for solution of MILPs, so this approach to obtaining a dual function is not likely to be tractable.

Although we cannot easily obtain the value function itself, we may be able to obtain an approximation that bounds the value function from below. In other words, we may be able to construct a function $F : \mathbb{R}^m \to \mathbb{R}$ that satisfies $F(d) \leq z(d)$ for all $d \in \mathbb{R}^m$. Given that we can do this, the question arises exactly how to select such a function from among the possible alternatives. A sensible method is to choose one that provides the best bound for the current right-hand side b. This results a dual suggested by Wolsey [4]:

$$\begin{aligned} D &= \max_{F:\mathbb{R}^m \to \mathbb{R}} \{F(b) \mid F(d) \leq z(d), d \in \mathbb{R}^m\}. && (6) \\ &= \max_{F:\mathbb{R}^m \to \mathbb{R}} \{F(b) \mid F(Ax) \leq cx, && (7) \\ &\qquad x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}, x \geq 0\} \end{aligned}$$

Note that this dual is an optimization problem over the space of all functions, in the same spirit as (4). However, it produces a dual function that can provide a valid bound for any right-hand side d.

An optimal solution F^* to this dual is a dual function that agrees with the value function at b. For any such optimal solution and any new right-hand side d, $F^*(d)$ is a lower bound on $z(d)$. However, it is clear that not all optimal solutions provide the same bound for a given vector d. 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. It is an open question whether (6) in fact provides the best criterion for selecting a dual function or whether it is possible to compute a dual function guaranteed to produce "reasonable" bounds within a specified neighborhood of b.

3. Obtaining Dual Solutions: There are a wide range of known procedures for obtaining solutions to (6). These will be reviewed in a forthcoming paper on integer programming duality [5] and will not be discussed at length here. We do, however, consider two aspects of this question that are directly relevant to our work. First, we consider whether it is possible to restrict the class of functions considered in (6) in some reasonable way. If we restrict F to be linear, then (6) reduces to $D = \max\{ub \mid uA \leq c\}$, 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 [6] showed that restricting to the class of convex functions still results in the same dual function.

In a series of papers, Johnson [7, 8, 9] and later Jeroslow [10] 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 (6) in the pure integer case as the *subadditive dual*

$$D = \max\{F(b) \mid F(a^i) \leq c_i, F \text{ subadditive}\}, \qquad (8)$$

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 and Jeroslow [11] subsequently showed that the value function can always be extended to a subadditive function defined on all of $\mathbb{R}^n$ [11]. In [10, 9, 12, 1], authors 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.

Our goal is to develop procedures that integrate well with current computational practice, so we next consider methods for producing dual information as a byproduct of the branch-and-bound and branch-and-cut algorithms. We restrict ourselves first 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.

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

For linear programs solved using the simplex algorithm, a dual function can be obtained by constructing an *optimal basis*, which is a nonsingular, square 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 [13]. Consider a partition of $\mathcal{P}$ into the subpolyhedra $\mathcal{P}_1, \ldots, \mathcal{P}_s$ in such a way that $\Pi \subseteq \cup_{i=1}^{s} \mathcal{P}_i$ and assume that these subpolyhedra are nonempty. Let LP$_i$ be the linear program $\min_{x \in \mathcal{P}_i} cx$ associated with the subpolyhedron $\mathcal{P}_i$. The usual optimality conditions for branch and bound are captured formally in the following observation.

Observation 1 *Let B^i be an optimal basis for LP$_i$. Let*

$$U = \min\{c_{B^i}(B^i)^{-1}b + \beta_i \mid 1 \leq i \leq s, \hat{x}^i \in \Pi\} \quad (9)$$

and

$$L = \min\{c_{B^i}(B^i)^{-1}b + \beta_i \mid 1 \leq i \leq s\}, \qquad (10)$$

where β represents the constant factors associated with the nonbasic variables fixed at nonzero bounds and $\hat{x}^i$ is the basic feasible solution corresponding to basis B^i. If $U = L$, then $P = U$ and for each $1 \leq j \leq s$ such that $\hat{x}^j \in \Pi$ and $c_{B^j}(B^j)^{-1}b + \beta_j = P$, $\hat{x}^j$ 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 (6) when it is made real-valued as before:

$$L(d) = \min\{c_{B^i}(B^i)^{-1}d + \beta_i \mid 1 \leq i \leq s\}. \qquad (11)$$

Unfortunately, this function is not subadditive and is therefore not a feasible solution to (8). However, it is feasible for (6) and can still be used quite effectively for sensitivity analysis and warm starting. Most importantly, it can be readily computed from data produced as a byproduct 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 (11) can also be extended to account for empty subpolyhedra. We are currently working on extensions of this entire framework to the more commonly employed branch-and-cut algorithm.

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 the optimality conditions described above and is similar in spirit to a scheme suggested by Roodman for Balas' additive algorithm [14]. 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 (or a subtree of such a tree, see Section 5), 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 given 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 discuss the construction of a proper warm start tree.

In addition to a warm start tree, we may also provide additional data 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 MILP solver framework, which is part of the COIN-OR software suite and can be accessed through SYMPHONY's native interface or 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, with 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. SYMPHONY can be stopped at any time and a warm start tree saved, then reloaded for later use. Details of the use and implementation of these methods can be found in the SYMPHONY 5.0 User's Manual [15] and in a recent proceedings paper [16].

SYMPHONY implements a standard branch-and-cut approach for solving MILP instances. A priority queue of candidate subproblems corresponding to the leaf nodes of a dynamically evolving branch-and-bound tree represents the current partition of the original problem, as well as the list of subproblems to be processed. The algorithm terminates when this queue is empty or when another pre-defined condition is satisfied. If the user instructs SYMPHONY to keep information to be used for later warm starting, a warm start data structure is created to store the description of the tree, as well as the other auxiliary data needed to restart the computation. This description contains complete information about the subproblem corresponding to each node in the search tree, including the branching decisions that led to the creation of the subproblem, the list of active variables and constraints, and warm start information for the subproblem itself (which is a linear program). In addition, other relevant information regarding the status of the computation is recorded, such as the current bounds and best feasible solution found so far. Using the warm start class, the user can save a warm start to disk, read one from disk, or restart the computation from any warm start after modifying parameters or the problem data itself. This allows the user to easily implement periodic check pointing, design dynamic algorithms in which the parameters are modified after the gap reaches a certain threshold, or to modify problem data during the solution process.

The most straightforward use of the warm start class is to restart the solver after modifying problem parameters. The solution process can be interrupted with a satisfied condition and parameters of the algorithm changed. For instance, after a predefined time or node limit or a targeted duality gap is reached, one may want to change the branching rule or search strategy. In this case, the master module automatically records the warm start information resulting from the last solve call and restarts from that checkpoint if a subsequent solve call is made, unless external warm start information is loaded manually.

The situation is more challenging if the user modifies problem data in between calls to the solver. In this case, SYMPHONY must make corresponding modifications to the leaves of the current search tree to allow execution of the algorithm to continue. Once the problem data is changed, the current tree may specify too fine a partition with which to restart the computation. SYMPHONY contains a number of ad hoc rules by which a subtree of the tree computed during the initial solve call can be used to determine a coarser partition of the solution space (see Section 4). The user can currently choose to take

1. the first k nodes generated during the initial solve,
2. all nodes above level k,
3. the first $t\%$ nodes generated, or
4. all nodes above level rl, where l is the depth of the tree and r is a parameter between 0 and 1.

For each option, all nodes but those in the chosen subtree are discarded and the computation is warm started from this given warm start tree. Each leaf node, regardless of its status after termination of the previous solve call, must be inserted into the queue of candidate nodes and reprocessed with the changed rim vectors. After this reprocessing, the computation can continue as usual.

6. Computational Results: We have performed a variety of tests with a selected subset of the standard test set for mixed-integer linear programming, MIPLIB3 [17]. In each of the following tables, the horizontal axis represents the individual instances, whereas the vertical axis represents the solution times. Tables 1 and 2 show the results of re-solving each instance from a warm start after perturbing a random subset of objective coefficients of random size. In each case, the warm start tree used was obtained by taking all nodes above level rl (rule 4 above). In Table 1, for instance, $r = 0$ means that the problem was solved from scratch (no warm start), $r = .5$ means the warm start tree consisted of the nodes whose level was less than half the total depth of the tree from the initial solve, and $r = 1$ means the complete tree from the initial solve was used. Tables 3 and 4 consist of the results of warm starting with $r = 1$ after modifying a random subset of objective coefficients with fixed size. Table 5 shows the results of warm starting with $r = 1$ after perturbing a random subset of right-hand side coefficients. In Table 6, a single knapsack problem with right-hand side b was re-solved with modified right-hand sides between $b/2$ and $3b/2$. For each right-hand-side, warm starting was used with $r = .25$.

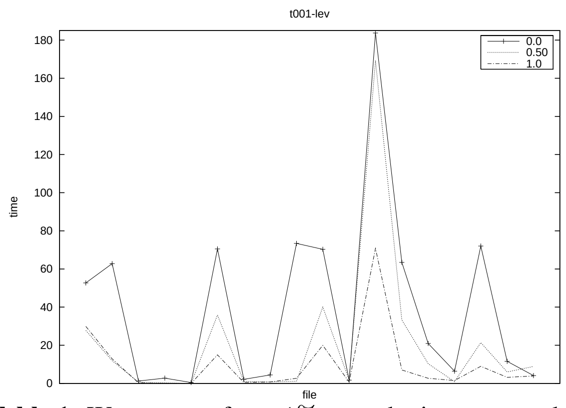


Table 1: Warm start after a 1% perturbation on a random subset of objective coefficients with $r \in \{0, .5, 1\}$.

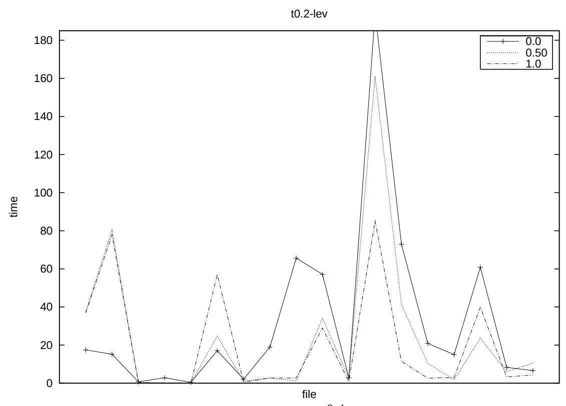


Table 2: Warm start after a 20% perturbation on a random subset of objective coefficients with $r \in \{0, .5, 1\}$.

These results show that for modifications of relatively small magnitude, warm starting yields good results on this test set. However, its efficiency tends to decrease as the magnitude of the perturbation increases. Comparing the behavior of warm starting for different values of r, we see that, although the option of warm starting with the complete tree seems to yield the best results overall, this is not true in in all cases. For the knapsack instance, $r = .25$ seemed to yield the best performance.

7. Sample Applications: The ability to resolve after modifying problem data is particularly important for applications in which a sequence of related MILP problems must be solved. To illustrate this, we have implemented algorithms for solving 2-stage stochastic integer programs (from [18]) and bicriteria integer programs (from [19]). Below, we will briefly discuss these algorithms and then present computational results.

7.1. Stochastic Programming: In [18], the authors used a Lagrangian-based dual decomposition algorithm to solve 2-stage stochastic pure integer programs. First, Black: without warm start, White: with warm start

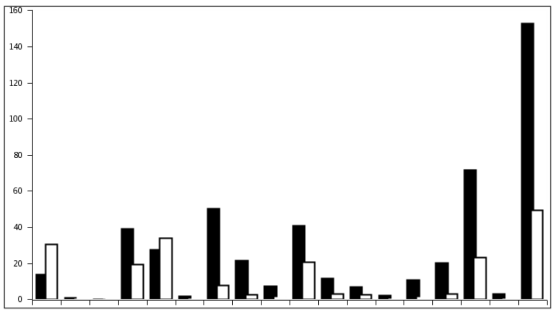


Table 3: Warm start after 10% perturbation of a random subset of objective coefficients of size $0.1n$.

Black: without warm start, White: with warm start

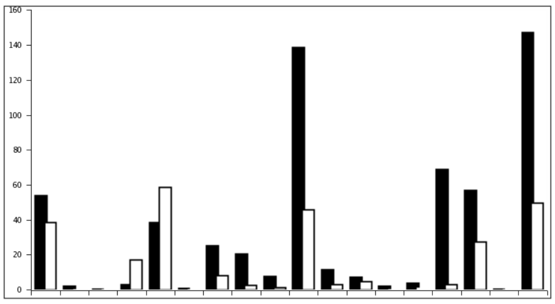


Table 4: Warm start after 10% perturbation of a random subset of objective coefficients of size $0.2n$.

Black: without warm start, White: with warm start

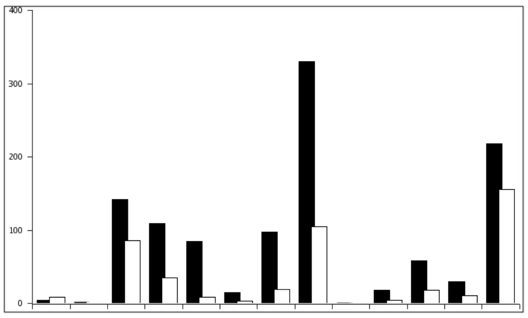


Table 5: Warm start after 20% perturbation of a random subset of right-hand side coefficients of size $0.1m$.

Black: without warm start, White: with warm start

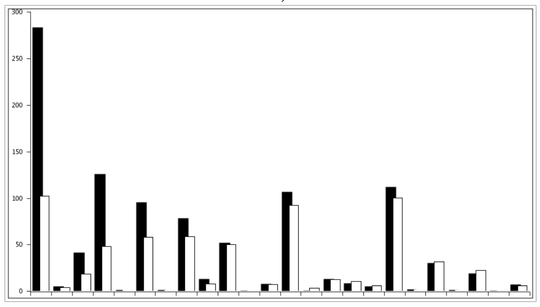


Table 6: Warm start after perturbing the right-hand side b of a knapsack problem.

consider the following two stage stochastic programming instance with fixed, relatively complete, integer recourse:

$$v = \min\{cx + Q(x) : Ax \geq b,\ x \in X\} \tag{12}$$

where $Q(x) = E_\xi[\theta(h(\xi) - T(\xi)x)]$ is the expected recourse value with $\theta(w) = \min\{q(\xi)y : Wy \geq w, y \in Y\}$. Here, c, b, A, W are known vectors and matrices, the vector ξ is a random variable defined on some probability space and for each ξ, the vectors $q(\xi), h(\xi)$ and the technology matrix $T(\xi)$ have appropriate dimensions. The sets X and Y denote the restriction that all variables $x \in X$ and $y \in Y$ are nonnegative, while some of them are integer or binary. For each ξ, a *scenario* is the realization of the random variable $(q(\xi), h(\xi), T(\xi))$.

Let the number of scenarios be k. Scenario j is denoted by $(q^j(\xi), h^j(\xi), T^j(\xi))$ and occurs with probability p_j, for $j = 1, ..., k$. If we define

$$\begin{aligned} S^j := \{ & (x, y^j) : Ax \geq b, x \in X, \\ & T^j x + Wy^j \geq h^j, y^j \in Y\}, \end{aligned}$$

then the deterministic equivalent of the problem can be written as

$$v = \min\{cx + \sum_j p^j q^j y^j : (x, y^j) \in S^j, j = 1, ..., k\}. \tag{13}$$

The algorithm in [18] is based on *scenario decomposition*, that is, we introduce copies of the first stage variables $x^1, ...x^k$ and rewrite the last equation as

$$\begin{aligned} v \;=\; & \min\{\sum_j p^j(cx^j + q^j y^j) : (x^j, y^j) \in S^j, \\ & j = 1, \ldots, k,\ x^1 = x^2 = ... = x^k\}. \end{aligned}$$

The last set of equality conditions on these copies is called the *non-anticipativity constraint* and states that the first stage decision should be independent of the second stage outcome. For convenience, the non-anticipativity constraint will be represented in matrix form by

$$\sum_j H^j x^j = 0, \tag{14}$$

where the matrices H^j, $j = 1, \ldots, k$ are defined appropriately.

Introducing the penalty vector u, the Lagrangian relaxation with respect to the non-anticipativity condition is the problem of finding $x^j, y^j, j = 1, \ldots, k$ such that

$$g(u) = \min\ \{\sum_j L_j(x^j, y^j, u) : (x^j, y^j) \in S^j\} \tag{15}$$

where $L_j(x^j, y^j, u) = p^j(cx^j + q^j y^j) + u(H^j x^j), j = 1, ..., k$. The problem

$$D = \max_{u \in \mathbb{R}^m} g(u) \tag{16}$$

Problem	Tree Size Without WS	Tree Size With WS	% Gap Without WS	% Gap With WS	CPU Without WS	CPU With WS
storm8	1	1	-	-	14.75	8.71
storm27	5	5	-	-	69.48	48.99
storm125	3	3	-	-	322.58	176.88
LandS27	71	69	-	-	6.50	4.99
LandS125	37	29	-	-	15.72	12.72
LandS216	39	35	-	-	30.59	24.80
dcap233_200	39	61	-	-	256.19	120.86
dcap233_300	111	89	0.387	-	1672.48	498.14
dcap233_500	21	36	24.701	14.831	1003	1004
dcap243_200	37	53	0.622	0.485	1244.17	1202.75
dcap243_300	64	220	0.0691	0.0461	1140.12	1150.35
dcap243_500	29	113	0.357	0.186	1219.17	1200.57
sizes3	225	165	-	-	789.71	219.92
sizes5	345	241	-	-	964.60	691.98
sizes10	241	429	0.104	0.0436	1671.25	1666.75

Table 7: Results of using warm starting to solve stochastic integer programs.

is a dual problem that provides a valid bound on the optimal solution value for the original problem. A nice feature of this dual problem is that for any u, we can separate the problem into subproblems for each scenario

$$g(u) = \sum_j g_j(u), \tag{17}$$

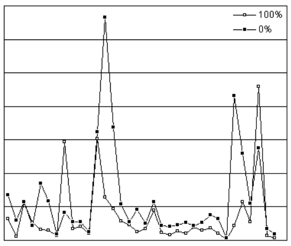
where for each $j = 1, \ldots, k$, $g_j(u) = \min\{L_j(x^j, y^j, u) : (x^j, y^j) \in S^j\}$ is a MILP. In [18], the authors show how to embed this bounding procedure into a branch-and-bound algorithm for solving the original problem.

We used a very straightforward implementation of the subgradient algorithm to solve each bounding problem. This requires solving a sequence of related MILPs. At the root node, the l^{th} iteration of subgradient optimization requires solving the problems $g_j(u^l) = \min \{L_j(x^j, y^j, u^l) : (x^j, y^j) \in S^j\}$, $j = 1, \ldots, r$. In the next iteration, the problem of evaluating $g_j(u^{l+1})$, for a given j, will differ from that of evaluating $g_j(u^l)$ only in the objective function. We used SYMPHONY to solve the subproblems with and without warm starting from one iteration to the next. SUTIL ([20]) was used to read in the instances. Table 7 shows the results of solving a set of 2-stage stochastic programs from [21, 22, 23]. The presence of a gap indicates that the problem was not solved to within the gap tolerance in the time limit. Although the running times are not competitive overall because of the slow convergence of our subgradient algorithm, one can clearly see the improvement arising from the use of warm starting.

7.2. Bicriteria Problems: A *bicriteria MILP* can be generalized by

$$\begin{array}{rl} \text{vmin} & [cx, qx] \\ \text{s.t} & x \in \Pi \end{array} . \tag{18}$$

Solving (18) is the problem of generating the set of *efficient* solutions $\mathcal{E} = \{x_1, ..., x_k\}$, where for any $\bar{x} \in \mathcal{E}$ there does not exist a second distinct feasible solution $\hat{x}$



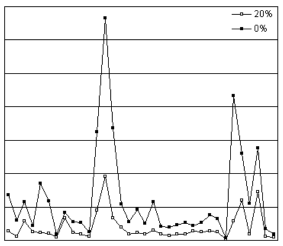


Table 8: Results of using warm starting to solve bicriteria optimization problems.

such that $c\hat{x} \leq c\bar{x}$ and $q\hat{x} \leq q\bar{x}$ and at least one inequality is strict. If both of the inequalities are strict, then $\bar{x}$ is called *strongly efficient*. The goal is to enumerate the set of all *Pareto outcomes* $\mathcal{R} = \{(cx, qx) \mid x \in \mathcal{E}\}$.

An extreme point of the lower envelope of $conv(\mathcal{R})$ is called a *supported outcome* and can be obtained by solving the *weighted sum* problem

$$\min_{x \in \Pi}((1-\phi)c + \phi q)x \tag{19}$$

where $\phi \in [0, 1]$. Clearly, the set of all supported outcomes can be obtained by solving (19) for all $\phi \in [0, 1]$.

On the other hand, the complete set of Pareto outcomes can be obtained by considering the *weighted Chebyshev norms* (WCN). Assume that $\mathcal{R}$ is *uniformly dominant* (that is, if all efficient solutions are strong). Let x^c be a solution to a weighted sum problem with $\phi = 0$ and x^q be a solution with $\phi = 1$ (these are called the *utopia* solution values). Then, for a given $\phi \in [0, 1]$, the WCN problem is

$$\min_{x \in \Pi} \max\{(1-\phi)(cx - cx^c), \phi(qx - qx^q)\} \tag{20}$$

and any solution to this problem corresponds to a Pareto outcome. Note that (20) can easily be linearized by introducing an additional variable and $\mathcal{E}$ can be produced by solving this MILP for all $\phi \in [0, 1]$. Such an algorithm was recently presented by [19]. We will not go into the details here, but note that the authors generate a set of candidate values for ϕ and solve a corresponding sequence of MILP instances.

SYMPHONY has an implementation of this basic algorithm. We have added the warm starting capability to the supported outcome generator, since in this case only the objective varies from one MILP instance to the next and generated valid inequalities also remain valid for all instances. We used the bicriteria solver to analyze instances of the Capacitated Network Routing Problem of size 15 reported in [19]. Because of the number of related subproblems being solved, there are a wide range of possible warm starting strategies. For this experiment, we kept the branch-and-cut trees of all instances solved so far and chose the warm start tree for a new instance to be derived from that of the previously solved subproblem with the closest ϕ value. We constructed the warm start tree by selecting the first $t\%$ of the nodes generated during the initial solution procedure for the chosen instance (rule 3 from Section 5). In Table 8, the first chart compares the results for t equal to 0 and 100, whereas the second chart compares the results for values of t equal to 0 and 20. The results with $t = 20$ show a clear and marked improvement over those with $t = 0$. Results with $t = 100$ show a noticeable effect in some cases, but an overall improvement.

8. Conclusions and Future Work: We have outlined a theory and methodology for warm starting computations that can be integrated with the branch-and-bound algorithm. This work lays the foundation for more significant advances, both theoretical and computational, to follow. The next step is to extend these techniques to the more sophisticated variants of branch and bound employed by most modern solvers, The implementation of such techniques is more involved, but we are currently working out the details of such methods. A number of questions still remain regarding the most efficient ways to employ these techniques in practice, but our preliminary results indicate that there are important potential gains to be made by developing these ideas further.

9. References:

[1] L. Wolsey, "Integer programming duality: Price functions and sensitivity analaysis," *Mathematical Programming*, vol. 20, pp. 173–195, 1981.

[2] H. Greenberg, "Mathematical programming glossary," http://carbon.cudenver.edu/~hgreenbe/glossary/index.php.

[3] C. Blair and R. Jeroslow, "The value function of an integer program," *Mathematical Programming*, vol. 23, pp. 237–273, 1982.

[4] L. Wolsey, "The b-hull of an integer program," *London School of Economics and CORE*, 1979.

[5] M. Güzelsoy and T. Ralphs, "Integer programming duality," Lehigh University Industrial and Systems Engineering, Tech. Rep., 2006.

[6] R. Jeroslow, "Minimal inequalities," *Mathematical Programming*, vol. 17, pp. 1–15, 1979.

[7] E. Johnson, "On the group problem for mixed integer programming," *Mathematical Programming Study*, vol. 2, pp. 137–179, 1974.

[8] ——, "Cyclic groups, cutting planes and shortest paths," *in Mathematical Programming, T.C. Hu and S.M. Robinson (eds.) Academic Press, New York, NY*, pp. 185–211, 1973.

[9] ——, "On the group problem and a subadditive approach to integer programming," *Annals of Discreet Mathematics*, vol. 5, pp. 97–112, 1979.

[10] R. Jeroslow, "Cutting plane theory: Algebraic methods," *Discrete Mathematics*, vol. 23, pp. 121–150, 1978.

[11] C. Blair and R. Jeroslow, "The value function of a mixed integer program: I," *Discrete Mathematics*, vol. 19, pp. 121–138, 1977.

[12] A. Bachem and R. Schrader, "Minimal equalities and subadditive duality," *Siam J. on Control and Optimization*, vol. 18, no. 4, pp. 437–443, 1980.

[13] F. Granot and J. Skorin-Kapov, "Some proximity and sensitivity results in quadratic integer programming," *Mathematical Programming*, vol. 47, pp. 259–268, 1990.

[14] G. Roodman, "Postoptimality analysis in zero-one programming by implicit enumeration," *Naval Research Logistics Quarterly*, vol. 19, pp. 435–447, 1972.

[15] T. Ralphs, "SYMPHONY Version 5.0 user's manual," www.BranchAndCut.org/SYMPHONY, Lehigh University Industrial and Systems Engineering, Technical Report 04T-011, 2004.

[16] T. Ralphs and M. Guzelsoy, "The SYMPHONY callable library for mixed-integer linear programming," in *Proceedings of the Ninth INFORMS Computing Society Conference*, 2005, pp. 61–76.

[17] R. Bixby, E. Boyd, and R. Indovina, "MIPLIB: A test set of mixed integer programming problems," *SIAM News*, vol. 25, p. 16, 1992.

[18] C. Caroe and R. Schultz, "Dual decomposition in stochastic integer programming," *Operations Research Letters*, no. 24, pp. 37–54, 1999.

[19] T. Ralphs, M. Saltzman, and M. Wiecek, "An improved algorithm for biobjective integer programming," 2006, to appear in Annals of Operations Research.

[20] J. Linderoth, "SUTIL, Available from http://coral.ie.lehigh.edu/sutil/index.html," 2006.

[21] S. Ahmed, 2004, SIPLIB, Available from http://www.isye.gatech.edu/~sahmed/siplib.

[22] A. Felt, "Stochastic linear programming data sets," 2004, available from http://www.uwsp.edu/math/afelt/slptestset.html.

[23] D. Holmes, "Stochastic linear programming data sets," 2004, available from http://users.iems.nwu.edu/~jrbirge/html/dholmes/post.html.