Fenchel Cutting Planes
for Linear Integer Programming Problems

E. Andrew Boyd$^1$

TR90-20

$^1$This work was sponsored in part by NSF grant ECS-8809053.
Abstract

The author recently introduced a class of cutting planes for integer programs called Fenchel cuts which distinguish themselves from more conventional cuts in that they are generated by directly seeking to solve the separation problem rather than through the use of explicit knowledge of the polyhedral structure of the integer program. An algorithm for generating Fenchel cuts is presented and described in detail for the separation problem associated with knapsack polyhedra. The paper concludes by using this algorithm to provably optimize a linear function over the intersection of the knapsack polyhedra defined by individual constraints in a collection of integer programs first introduced by Crowder, Johnson, and Padberg.
1 Introduction

In 1983 Crowder, Johnson, and Padberg won the highest honor offered by the Operations Research Society of America — the Lanchester prize — for some remarkable work solving integer programs. Using cutting planes they were able to solve all but one of a collection of integer programs to optimality in under 15 minutes even though “most [of the problems] were originally considered not amenable to exact solution in economically feasible computation times” ([3], p. 828). The largest most difficult problem required only slightly less than an hour to solve. Their work sparked renewed interest in the possibility of solving large integer programs with no special structure to optimality.

While the work by Crowder, Johnson, and Padberg represents an important success story, successful research on cutting plane methods extends far beyond this single work. Phenomenal computational results have also been achieved on a number of important specialized problem classes using cutting planes. Early work by Grötschel [7], Padberg and Rinaldi [16], and others led to the solution of much larger traveling salesman problems than had ever been solved previously and ongoing work on this problem continues to yield substantial performance improvements. Cutting plane methods have even begun to find their way into publically available general purpose algorithms such as IBM’s OSL and the Georgia Tech’s GT-MIO.

The most fundamental problem arising in the use of cutting planes to solve integer programs is the separation problem — the problem of finding an inequality that is valid for the polyhedron defined by the convex hull of all feasible integer points but that is violated by the optimal solution to the linear programming relaxation of the problem. Commonly, separation algorithms are devised for known classes of valid inequalities with good theoretical characteristics, usually facet classes. In practice, good separation algorithms are far more scarce than known classes of cutting planes and in general appear to be more elusive.

The author recently introduced a class of cutting planes called Fenchel cuts which differ from more conventional cutting planes in that they focus directly on the separation problem without
reference to an underlying class of cutting planes. Fenchel cuts are generated by maximizing a piecewise linear concave function \( v(\lambda) \). It can be shown that Fenchel cuts are the deepest cuts that can be generated for a problem in a well defined sense and that if the maximum value of \( v(\lambda) \) is nonpositive then no cutting plane exists. Fenchel cuts and their relation to another class of cutting planes associated with Lagrangian relaxation are described in [2].

In this paper we address the problem of maximizing the function \( v(\lambda) \) associated with Fenchel cut generation for knapsack polyhedra. An algorithm for generating Fenchel cuts is presented and described in detail for the separation problem associated with knapsack polyhedra. The paper concludes by provably optimizing a linear function over the intersection of the knapsack polyhedra defined by the constraints of the integer programs used by Crowder, Johnson, and Padberg in [3] — a result that has not been achieved prior to this paper.

2 Fenchel Cuts

The underlying theory of Fenchel cuts is developed in [2]. In this section, we outline the aspects of this theory necessary for the developments presented in this paper.

Consider the following problem.

\[
\begin{align*}
\text{max} & \quad cx \\
\text{s.t.} & \quad Qx = q \\
& \quad Rx \leq r \\
& \quad Ax = a \\
& \quad Bx \leq b \\
& \quad x \text{ integer}
\end{align*}
\]

Let \( F = \{x : Ax = a, Bx \leq b, x \text{ integer}\} \) and let \( P_F \) denote the convex hull of \( F \). Further, let \( \hat{x} \) be feasible for the constraints of (P) with the exception of the integrality restriction. Conceptually, \( \hat{x} \) can be thought of as a point generated by solving the linear programming relaxation of (P). The cut
generation procedure to be described seeks an inequality that is not satisfied by \( \hat{x} \) but that contains \( P_F \) and therefore the feasible region of \( (P) \).

As but one example, \( F \) might be defined by upper and lower bound constraints on the variables together with a single constraint taken from the original problem \( (P) \). This collection of relaxations \( F \), one defined by each row of \( (P) \), is exactly the collection of relaxations used by Crowder, Johnson, and Padberg in [3].

Let the rows of \( D \) span the nullspace of \( A \). We define \( f(\lambda) \) and \( v(\lambda) \) as follows.

\[
f(\lambda) = \max\{\lambda^T D x : x \in P_F\}
\]

\[
v(\lambda) = \lambda^T D \hat{x} - f(\lambda)
\]

The following is proved in [2].

**Theorem 1** There exists a value \( \lambda \) for which \( v(\lambda) > 0 \) if and only if there exists a hyperplane \( \lambda^T D x \leq f(\lambda) \) separating \( \hat{x} \) from \( P_F \).

The practical implication of Theorem 1 is that the question of whether or not there exists a cutting plane separating \( \hat{x} \) from \( P_F \) can be answered by investigating whether or not the function \( v(\lambda) \) achieves a positive value. For any fixed value of \( \lambda \) the inequality

\[
\lambda^T D x \leq f(\lambda)
\]

is valid for \( P_F \), and this inequality separates \( P_F \) from \( \hat{x} \) if and only if \( v(\lambda) > 0 \). Due to connections with Fenchel duality such cuts were deemed Fenchel cuts.

The following theorem, also proved in [2], makes note of important theoretical properties that simplify finding values of \( \lambda \) for which \( v(\lambda) > 0 \).

**Theorem 2** The function \( v(\lambda) \) is piecewise linear and concave. Specifically, \( v(\lambda) \) can be expressed as

\[
v(\lambda) = \min\{\lambda^T D x^1 - \lambda^T D \hat{x} : x^1 \in E(P_F)\}
\]
where $E(\mathcal{P})$ is the set of extreme points of $\mathcal{P}$.

The following observation follows directly from the definition of $v(\lambda)$.

**Observation 1** For any scalar $\omega$, $v(\omega \lambda) = \omega v(\lambda)$.

The immediate implication of this observation is that if $v(\lambda)$ achieves a positive value it achieves a positive value on any full dimensional set containing the origin in its strict interior. In fact, it is not difficult to verify the following observation.

**Observation 2** $v(\lambda)/\|\lambda D\|$ is the distance from $\hat{x}$ to the plane $\lambda Dz = f(\lambda)$ when $\lambda D\hat{x} \leq f(\lambda)$ separates $\hat{x}$ and $\mathcal{P}$, and the negative of this distance when it does not.

Thus, solving the maximization problem

$$\max \quad v(\lambda)$$

$$\text{s.t.} \quad \|\lambda D\| \leq 1$$

generates the deepest cut separating a point $\hat{x}$ from $\mathcal{P}$. In practice, it is easier to attempt to maximize $v(\lambda)$ on a linear domain. Further, through the appropriate choice of domain it is possible to affect the polyhedral characteristics of the generated cut. This use of the domain is discussed in more detail in Section 4.

In summary, Fenchel cuts are generated by seeking to maximize the function $v(\lambda)$ on any domain with appropriate characteristics. Any value of $\lambda$ with $v(\lambda) > 0$ corresponds to a cutting plane, and if the maximum value of $v(\lambda)$ is zero or less then this represents a proof that there exists no cutting plane separating $\hat{x}$ from $\mathcal{P}$.

The way in which Fenchel cuts are generated is fundamentally different than the way in which most cutting planes are generated and as such they provide unique theoretical and computational possibilities. Most cutting planes are derived from classes of theoretically studied facets for given problems and the effectiveness of these cutting planes is governed by the existence of good separation algorithms for generating violated cuts. In contrast, separation is the essence of Fenchel cuts.
3 Generating Fenchel Cuts

The major computational task in the generation of Fenchel cuts is that of finding a value of \( \lambda \) for which \( v(\lambda) > 0 \), and preferably a value for which \( v(\lambda) \) is maximized. The piecewise linear concavity of \( v(\lambda) \) together with the ability to choose a convex domain provide necessary theoretical properties for maximizing \( v(\lambda) \). In addition, as with the maximization of Lagrangian dual functions, it is possible to prove that a subgradient of \( v(\lambda) \) is generated whenever a value of \( v(\lambda) \) is calculated. It is therefore possible to seek to maximize \( v(\lambda) \) using subgradient techniques, generalized programming, or other well-established nondifferentiable optimization techniques.

In practice, however, subgradient techniques and generalized programming very commonly demonstrate extremely poor convergence. Often this convergence is sufficiently bad to render these methods effectively useless. (The ineffectiveness of these methods is discussed in Section 6 and served as a primary motivation for the work described in this paper.) Further, ascent algorithms based on subgradient information do not tend to generate extreme points of the hypograph of \( v(\lambda) \) while generalized programming algorithms do not do so unless run to optimality. Fenchel cuts generated by these algorithms thus fail to have the desirable polyhedral properties (described in the following section) demonstrated by Fenchel cuts corresponding to these extreme points.

The practical motivation for the use of subgradient techniques and generalized programming is that they require only an oracle that for any value of \( \lambda \) returns \( v(\lambda) \) and an associated extreme point \( x^i \) of \( P_F \) defining \( v(\lambda) \). An algorithm for optimizing a linear function on \( P_F \) serves this purpose. In the case of the knapsack polyhedron, however, it is possible to parametrically optimize a linear function on \( P_F \). This provides a stronger oracle and makes it possible to develop a more efficient algorithm for maximizing \( v(\lambda) \) than algorithms based on subgradient techniques or generalized programming. We briefly describe the algorithm in general terms before considering its application to knapsack polyhedra.
Consider the following problem.

\[
\begin{align*}
\text{max} & \quad z \\
\text{s.t.} & \quad z \leq \lambda D\hat{x} - \lambda Dz^i \quad x^i \in E(P_F) \\
\lambda & \in \Lambda = \{\lambda : P\lambda \leq p\}
\end{align*}
\]

Here \( \Lambda \) is any set of constraints containing a full dimensional ball about the origin, such as \(-1 \leq \lambda \leq 1\). We refer to the constraints \( P\lambda \leq p \) as the \( \Lambda \) constraints and all remaining constraints as the \( X \) constraints. Conceptually, \((G)\) is nothing more than a linear program with maximizing value equal to the maximum value of \( v(\lambda) \) on the domain \( \Lambda \). The obvious practical difficulty with solving \((G)\) is, of course, that the \( X \) constraints are defined implicitly by the extreme points \( x^i \) of \( P_F \) and in most instances there are an exponential number of these constraints. Nonetheless, an ascent algorithm can be applied to solve \((G)\) if an oracle exists for solving the following problem.

**Problem FPARAM:** Let \([\bar{\lambda}, v(\bar{\lambda})]\) be a feasible vector for \((G)\), let \( S_X \) be a subset of the \( X \) constraints satisfied at equality by \([\bar{\lambda}, v(\bar{\lambda})]\), and let \( S_\Lambda \) be a subset of the \( \Lambda \) constraints satisfied at equality by \([\bar{\lambda}, v(\bar{\lambda})]\). Given a direction \([d, 1]\) such that \([\bar{\lambda}, v(\bar{\lambda})] + \theta[d, 1]\) satisfies all of the constraints \( S_X \) and \( S_\Lambda \) for all \( \theta \geq 0 \), find the largest value of \( \theta \) such that \([\bar{\lambda}, v(\bar{\lambda})] + \theta[d, 1]\) does not violate any of the constraints in \((G)\).

An algorithm for maximizing \( v(\lambda) \) that makes use of an oracle for solving problem FPARAM is presented in Figure 1. Although it is not immediately apparent, the algorithm presented in Figure 1 is exactly the primal simplex algorithm if \([\bar{\lambda}, v(\bar{\lambda})]\) is initially an extreme point of the polyhedron defined by \((G)\) and if the choice of the ascent direction in Step 2 is limited to edges of this polyhedron. A detailed description of this algorithm using the primal simplex interpretation can be used to formally establish that the procedure maximizes \( v(\lambda) \) after a finite number of iterations.

As described, the algorithm is somewhat more flexible than the primal simplex algorithm and instead of dwelling upon this interpretation we choose simply to make some instructive observations about the algorithm.
Step 1: **Initialize.** Choose a vector \([\lambda, v(\lambda)]\) feasible for (G), a subset \(S_X\) of the \(X\) constraints satisfied at equality by \([\lambda, v(\lambda)]\), and a subset \(S_A\) of the \(A\) constraints satisfied at equality by \([\lambda, v(\lambda)]\).

Step 2: Choose an ascent direction \([d, 1]\) such that \([\lambda, v(\lambda)] + \theta[d, 1]\) satisfies all of the constraints \(S_X\) and \(S_A\) for all \(\theta \geq 0\). If no such direction exists, stop; \([\lambda, v(\lambda)]\) is optimal for (G).

Step 3: Solve problem \(FPARAM\) for \(\theta\) and let \([\lambda, v(\lambda)] = [\lambda, v(\lambda)] + \theta[d, 1]\). Include the constraint that defined \(\theta\) in the appropriate set \(S_X\) or \(S_A\) and remove from \(S_X\) and \(S_A\) those constraints that are not satisfied at equality by the new \([\lambda, v(\lambda)]\). Go to Step 2.

Figure 1: Algorithm to Maximize \(v(\lambda)\)

The termination criterion given in Step 2 follows from the fact that the candidate solution \([\lambda, v(\lambda)]\) is feasible for (G) and satisfies optimality conditions for a relaxation of (G). The direction \([d, 1]\) chosen in Step 2 is an ascent direction for the relaxation of (G) defined by the constraints \(S_X\) and \(S_A\) but may not be a true ascent direction for the problem (G). Specifically, there may be \(X\) or \(A\) constraints that are candidates to be in the sets \(S_X\) and \(S_A\) that are not in these sets so that any positive step length moves outside the feasible region of (G). The algorithm for solving problem \(FPARAM\) in Step 3 answers the question of how long a step may be taken in the direction \([d, 1]\) without violating any of the constraints in problem (G). As just noted, this step length may be 0. The algorithm presented in Figure 1 can thus be seen to be an ascent algorithm which makes use of an oracle for generating step length.

Clearly, a good algorithm for solving problem \(FPARAM\) will not in general exist for an arbitrary subproblem polyhedron \(P_F\). However, as a general observation, the ability to optimize a linear function on \(P_F\) is indicative of a subproblem structure that will allow the problem \(FPARAM\) to be solved as well. This is the case for the knapsack polyhedron, and the following sections examine an instance of the algorithm presented in Figure 1 as applied to this polyhedron.
4 Fenchel Cuts in Integer Programming

As was mentioned at the outset of this paper Crowder, Johnson, and Padberg used the knapsack polyhedra defined by individual constraints of an integer program as a basis for generating cutting planes. Specifically, they considered the polyhedra $P_F$ defined by

$$P_F^i = \text{conv}\{x: a^i x \leq b, \quad 0 \leq x \leq 1, \quad x \text{ integer}\}$$

where $x$ is conformable to the $n$-vector $a^i$ of non-zero elements in constraint $i$. In this section we present a highly efficient algorithm for generating Fenchel cuts separating a point $\hat{x}$ from $P_F^i$. For simplicity of exposition we assume henceforth that $a^i > 0$ since this is easily achieved through complementing variables.

The algorithm attempts to generate cuts by seeking values of $\lambda$ for which $v(\lambda) > 0$. Before embarking on a discussion of the algorithm itself, however, there remain issues regarding the domain on which $v(\lambda)$ should be optimized. As noted in Observation 2, maximizing $v(\lambda)$ on the domain $\|\lambda D\| \leq 1$ has the attractive theoretical property of generating a hyperplane for $P_F^i$ that is as far as possible from $\hat{x}$. Further, if the rows of $D$ are chosen so that they are orthonormal the domain $\|\lambda D\| \leq 1$ reduces to simple domain $\|\lambda\| \leq 1$. Nonetheless, the nonlinearity is unattractive for obvious reasons, especially when Observation 1 presents the opportunity of choosing a linear domain. Linearly constrained domains also have some attractive theoretical properties which will be elaborated upon later in this section.

The specific form of the constraints defining $P_F^i$ also provides some simplification of potential domains.

**Theorem 3** Let $v(\lambda)$ be defined by $P_F^i$, assume $\hat{x} \geq 0$, and let $D = I$. There exists a $\lambda \geq 0$ which maximizes $v(\lambda)$ on the domain $\|\lambda\| \leq 1$. Further, if $\hat{x}_j = 0$ then there exists a $\lambda$ with $\lambda_j = 0$ which maximizes $v(\lambda)$ on the domain $\|\lambda\| \leq 1$.

**Proof.** Since all of the coefficients in the knapsack constraint defining the polyhedron $P_F^i$ are
positive by assumption, if \( y \in \mathcal{P}_F^i \) then for any \( x \geq 0, x \leq y \) it follows that \( x \in \mathcal{P}_F^i \). Suppose \( \bar{\lambda} \) maximizes \( v(\lambda) \) on the domain \( ||\lambda|| \leq 1 \) but that \( \bar{\lambda}_j < 0 \) for some index \( j \). Let \( \bar{z} \in \mathcal{P}_F^i \) be a maximizing value of \( x \) associated with \( \bar{\lambda} \). If \( \bar{z}_j > 0 \) then \( \bar{z}_j \in \mathcal{P}_F^i \) obtained from \( \bar{z} \) by setting \( \bar{z}_j = 0 \) has \( \bar{\lambda}_j > \bar{\lambda}_j \); that is, \( \bar{z} \) cannot be a maximizing value of \( x \) associated with \( \bar{\lambda} \). Thus, assume \( \bar{z}_j = 0 \) and consider \( \bar{\lambda}' \) formed from \( \bar{\lambda} \) by setting \( \bar{\lambda}_j = 0 \). Clearly, \( \bar{\lambda}' \) must be a maximizing value of \( x \) associated with \( \bar{\lambda}' \) as well as with \( \bar{\lambda} \). If not, then there exists an \( \bar{\lambda}' \in \mathcal{P}_F^i \) such that \( \bar{\lambda}' \bar{\lambda} \bar{\lambda}' = \bar{\lambda}' \bar{\lambda} \bar{\lambda}' \); that is, \( \bar{\lambda}' \) is not a maximizing value of \( x \) for \( \bar{\lambda} \). Thus, \( v(\bar{\lambda}) = \bar{\lambda}' \bar{\lambda} - \bar{\lambda}' \bar{\lambda} \bar{\lambda}' \leq \bar{\lambda}' \bar{\lambda} - \bar{\lambda}' \bar{\lambda}' = v(\bar{\lambda})' \). Using this technique to eliminate any \( \lambda_j < 0 \) it follows that there exists a value of \( \lambda \geq 0 \) which maximizes \( v(\lambda) \) on the domain \( ||\lambda|| \leq 1 \).

To complete the proof, let \( \bar{\lambda} \geq 0 \) maximize \( v(\lambda) \) on the domain \( ||\lambda|| \leq 1 \), let \( \bar{z} \in \mathcal{P}_F^i \) be a maximizing value of \( x \) associated with \( \bar{\lambda} \), and assume \( \bar{z}_j = 0 \) but \( \bar{\lambda}_j > 0 \). Let \( \bar{\lambda}' \) be \( \bar{\lambda} \) with \( \bar{\lambda}_j = 0 \). Since \( \bar{\lambda}_x \leq \bar{\lambda}_x \) is valid for \( \mathcal{P}_F^i \) and since \( \bar{\lambda} \geq 0 \) it follows that \( \bar{\lambda}' \bar{\lambda} \bar{\lambda}' \leq \bar{\lambda}_x \) is valid for all \( x \in \mathcal{P}_F^i \) although it may not be a face of \( \mathcal{P}_F^i \). Letting \( \bar{z}' \in \mathcal{P}_F^i \) be a maximizing value of \( x \) for \( \bar{\lambda}' \) it follows from the validity of \( \bar{\lambda}_x \leq \bar{\lambda}_x \) that \( \bar{\lambda}' \bar{\lambda}' \leq \bar{\lambda}_x \). Clearly, since \( \bar{\lambda}_x = \bar{\lambda}_x \) under the assumption that \( \bar{z}_j = 0 \), it follows that \( v(\bar{\lambda}') = \bar{\lambda}_x \bar{\lambda} - \bar{\lambda}_x \bar{\lambda}_x \leq \bar{\lambda}_x \bar{\lambda} - \bar{\lambda}_x \bar{\lambda}_x = v(\bar{\lambda}) \). Using this technique to eliminate any \( \lambda_j > 0 \) when \( \bar{z}_j = 0 \) it follows that if \( \bar{z}_j = 0 \) then there exists a \( \lambda \) with \( \lambda_j = 0 \) which maximizes \( v(\lambda) \) on the domain \( ||\lambda|| \leq 1 \). \( \square \)

Observation 1 yields the following corollary.

**Corollary 1** Let \( v(\lambda) \) be defined by \( \mathcal{P}_F^i \), assume \( \bar{\lambda} \geq 0 \), and let \( D = I \). Let \( \mathbb{R}^{n+} \) be the space corresponding to indices of \( \lambda \) for which \( \bar{\lambda}_i > 0 \) and let \( \Lambda \) be a set defined by the intersection of the positive orthant in \( \mathbb{R}^{n+} \) with any full dimensional body containing the origin in its strict interior.

If there exists a value of \( \lambda \) for which \( v(\lambda) > 0 \) then there exists a value of \( \lambda \in \Lambda \) for which \( v(\lambda) > 0 \).

The ability to restrict some values of \( \lambda \) to zero has an extremely important impact on finding values of \( \lambda \) for which \( v(\lambda) > 0 \). In practice, when \( \bar{\lambda} \) is a subvector of a solution to a larger linear program, many of the \( \bar{\lambda}_i \) may well be 0. This can serve to significantly reduce the dimension of the
space in which maximization of $v(\lambda)$ occurs and consequently reduce the time required to maximize $v(\lambda)$ dramatically. It is henceforth assumed that all $\lambda_i > 0$ recognizing that in general $P_f$ can simply be defined on this restricted space.

Corollary 1 clearly suggests that the constraints $\lambda_i \geq 0$ should be included in the definition of the domain $\Lambda$, while Observation 2 suggests that a collection of constraints approximating the unit sphere as it intersects the positive orthant would be reasonable. One appropriate collection of constraints is therefore the following.

$$\Lambda = \{\lambda : \sum_{i=1}^{n} \lambda_i \leq \beta, \ 0 \leq \lambda \leq 1\}$$

In this definition $\beta$ is any constant satisfying $0 < \beta \leq n$. It is important to realize that while Observation 2 provides motivation for approximating the unit sphere Corollary 1 makes it equally clear that a good approximation is not necessary to ensure the generation of a cutting plane if one exists. The simplicity of $\Lambda$ presents significant computational advantages over more complicated approximations to the unit sphere. Even more important, however, is that cuts generated using $\Lambda$ tend to have better polyhedral characteristics than even the unit sphere itself. The following theorem, which is true for arbitrary problems $(G)$, helps formalize this claim.

**Theorem 4** If $x_*$ is a unique optimal solution for $(G)$ then the Fenchel cut associated with $\Lambda$ defines a face of $P_F$ of dimension at least $n + 1 - |S_\Lambda|$, where $S_\Lambda$ is the set of $\Lambda$ constraints satisfied at equality by $x_*$. 

**Proof.** Let $S_X$ and $S_\Lambda$ denote the collection of $X$ and $\Lambda$ constraints satisfied at equality by $\Lambda$. Since $\Lambda$ is unique it follows that the normals of the constraints $S_X$ and $S_\Lambda$ must together span $\mathbb{R}^{n+1}$. Thus, the set $S_X$ must contain at least $n + 1 - |S_\Lambda|$ constraints $0 \leq \lambda(Dx_ - Dx^i) - z$ with linearly independent normal vectors $[Dx_ - Dx^i, -1]$. Further, the vectors $[-Dx^i, -1]$ are affinely independent since subtracting the constant vector $[Dx, 0]$ preserves affine independence. The vectors $Dx^i$ are therefore linearly independent implying the vectors $x^i$ must be affinely independent. Since
the $x^i$ satisfy the Fenchel cut associated with $\lambda$ at equality by virtue of the fact they are in $S_\lambda$, the result follows. \[\square\]

More general versions of Theorem 4 can be proved at the expense of somewhat more tedious proofs. Some care must be taken in interpreting this theorem. For example, whether or not increasing the number of $A$ constraints will increase the size of the set of active $A$ constraints at optimality is completely dependent on how these constraints are chosen. In particular, a more refined linear approximation to the unit sphere than provided by the constraints defining $\Lambda$ will not necessarily have a larger set $S_\lambda$. The following claim, which is really a converse of part of Theorem 3, provides evidence for one case where $|S_\lambda|$ is small.

Claim 1 Let $v(\lambda)$ be defined by $P_\lambda^f$, assume $\hat{x} \geq 0$, and let $D = I$. If $\hat{x} > 0$ then the value of $\lambda$ which maximizes $v(\lambda)$ on the domain $||\lambda|| \leq 1$ will generally have $\lambda_i > 0$.

Claim 1 is not precise, but both empirical evidence and an understanding of how Fenchel cuts relate to knapsack polyhedra support this observation. Given Observation 1, we can make the following claim as well.

Claim 2 Let $v(\lambda)$ be defined by $P_\lambda^f$, assume $\hat{x} \geq 0$, and let $D = I$. If $v(\lambda)$ is maximized on the domain $\Lambda$ with $\beta < 1$ then the associated Fenchel cut will tend to be a facet of the polyhedron $P_\lambda^f$ (restricted to the space for which $\hat{x} > 0$).

The argument for Claim 2 is, of course, that if all $\lambda_i > 0$ as proposed in Claim 1 then with $\beta < 1$ there is only one constraint in $\Lambda$ that can possibly be in $S_\lambda$. Thus, unique optimal solutions $\lambda$ would correspond to facet defining Fenchel cuts by Theorem 4.

Claim 2 raises the question of why the constraints $\lambda \leq 1$ are included in the definition of $\Lambda$ at all. The answer to this question is related to issues of degeneracy and is best answered in Section 6. In the end, while arguments have been presented for choosing to define $\Lambda$ as it has been defined, the best practical choice for $\Lambda$ is best determined empirically. The results of Section 6 support the choice of $\Lambda$. 

11
5 Generating Fenchel Cuts for Knapsack Polyhedra

In this section we develop an algorithm for solving the problem (G) defined by a knapsack polyhedron following the outline of the general algorithm given in Section 3. For the set \( A \) in (G) we use the set \( \tilde{A} \) defined in the previous section. For explanatory purposes it will prove useful to work with the following equality constrained version of (G) obtained by adding slack variables to the constraints.

\[
\begin{align*}
\max & \quad z \\
\text{s.t.} & \quad z - \lambda(D\bar{x} - D\tilde{x}^i) + s_i = 0 \\
& \quad \sum_{j=1}^n \lambda_j + s_0 = \beta \\
& \quad 0 \leq \lambda \leq 1
\end{align*}
\]

It will prove useful to refer to the constraint \( \sum_{j=1}^n \lambda_j + s_0 = \beta \) as the \( \beta \) constraint and to adhere to the tradition of referring to the constraints \( 0 \leq \lambda \leq 1 \) as simply the bounds and the remaining constraints as the constraints. We describe the algorithm for solving (G') by describing each of the three steps outlined in Section 3.

**Implementation of Step 1**

The set \( S_A \) is initialized to some subset of \( n \) of the bound constraints, with \( \bar{\lambda} \) chosen to be the point defined by the intersection of these constraints. The remaining constraint and the value \( v(\bar{\lambda}) \) are initialized by solving the problem FPARAM with an arbitrary direction. Since \( S_X \) is empty, the solution to this problem is an \( X \) constraint (from which the value \( v(\bar{\lambda}) \) can be calculated) and a value of 0 for \( \theta \). With \( S_X \) containing the returned constraint, \( S_X, S_A, \) and \( [\bar{\lambda}, v(\bar{\lambda})] \) are appropriately initialized for the algorithm to commence. It is easily verified that the point \( [\bar{\lambda}, v(\bar{\lambda})] \) is an extreme point of the polyhedron defined by the constraints of (G) and that the constraints comprising \( S_X \) and \( S_A \) have a linearly independent normals.
Implementation of Step 2

Consider the linear program

$$\max \ z$$

$$(H) \ s.t. \ S_X \ and \ S_A.$$ 

Since \( [\lambda, v(\lambda)] \) satisfies all of the \( S_X \) and \( S_A \) constraints at equality, Farkas' lemma can be used to demonstrate that either \( [\lambda, v(\lambda)] \) is optimal for this problem or that there exists a direction \( [d, 1] \) such that \( [\lambda, v(\lambda)] + \theta[d, 1] \) satisfies all of the constraints \( S_X \) and \( S_A \) for all \( \theta \geq 0 \). It follows that Step 2 can be completed by solving the linear program \( (H) \).

In practice, the need to invoke a linear program solver to solve \((H)\) each time Step 2 is performed is alleviated by maintaining sets \( S_X \) and \( S_A \) with special properties. In particular, the sets \( S_X \) and \( S_A \) are maintained so that together they contain a collection of \( n+1 \) constraints whose normals are linearly independent.

To appreciate the importance of this condition, consider how the linear program \( (H) \) is solved. \((H)\) is modified by appending slack variables to the constraints in \( S_X \) and to the \( \beta \) constraint if it is in \( S_A \) and these slack variables are restricted to be nonnegative. The bounds in \( S_A \) are treated explicitly as bound constraints and are not modified. In this form, simplex optimality conditions are achieved if all unbounded variables are basic, the reduced costs of all nonbasic variables at their upper bounds are nonnegative, and the reduced costs of all nonbasic variables at their lower bounds are nonpositive. A feasible solution for \((H)\) is \( [\lambda, v(\lambda)] \), and since this point satisfies all of the constraints in \( S_X \) and \( S_A \) at equality the slack variables and variables with bound constraints in \( S_A \) can be treated as nonbasic variables if the columns of the remaining variables \(- z \) and the variables \( \lambda_i \) not represented in \( S_A \) form a basis for the column space of \((H)\) in equality form. If \( S_X \) and \( S_A \) contain \( n+1 \) total constraints the total number of remaining variables is equal to the number of constraints in \((H)\). Further, it is easily verified that if the constraints in \( S_X \) and \( S_A \) have linearly independent normals then the columns corresponding to these remaining variables must be linearly independent. Thus, by maintaining the stated properties of \( S_X \) and \( S_A \) a basic feasible solution for
(H) is readily attainable.

A point of computational interest is related to the relative sizes of the sets $S_X$ and $S_A$. Letting $B$ be the basis matrix of (H) and $c_B$ the corresponding cost vector, dual variables $y$ can be determined in the usual way by solving $yB = c_B$ and the reduced costs of each of the nonbasic variables $j$ can be calculated as $c_j - ya^j$ where $a^j$ is the column of variable $j$ in the constraint matrix for (H). The dimension of $B$ and the $a^j$ is equal to the number of constraints in $S_X$ plus 1 if the $\beta$ constraint is in $S_A$. Thus, when the number of $S_X$ constraints is small as is the case in the early stages of the algorithm, the size of $B$ and the $a^j$ is correspondingly small therefore reducing the computational burden associated with Step 2.

Beyond providing sufficient conditions for quickly determining a feasible basis for (H), the stated restrictions on $S_X$ and $S_A$ also ensure that if the optimality of (H) is not demonstrated then a single pivot is sufficient to determine a direction of unboundedness in (H). This is easily seen to be the case since the basic variables are unbounded and each potential entering variable can have at most one bound represented in $S_A$.

As mentioned in Step 1, the constraints in $S_X$ and $S_A$ are initialized with the appropriate properties but it remains to demonstrate that these properties can be maintained throughout the algorithm. In point of fact, this is not difficult to do. The choice of ascent direction as described above is such that all but one of the constraints in $S_X$ and $S_A$ continue to be satisfied at equality when moving in this direction. In particular, if $[d, 1]$ is the direction of ascent then $[\lambda, v(\lambda)] + \theta[d, 1]$ satisfies the remaining $n-1$ constraints in $S_X$ and $S_A$ for all $\theta \geq 0$. Consider the constraint returned by solving problem FPARAM in Step 3. If the normal of this constraint could be expressed as a linear combination of the remaining $n-1$ constraints then $[\lambda, v(\lambda)] + \theta[d, 1]$ would necessarily satisfy this constraint for all $\theta \geq 0$ as well. However, this is a contradiction to the properties of the constraint returned by an algorithm solving problem FPARAM. It follows that the $n-1$ constraints together with the constraint determined by solving problem FPARAM have the properties required by the sets $S_X$ and $S_A$. 

14
In summary, given the stated properties on $S_X$ and $S_A$, Step 2 of the algorithm can be completed quite efficiently and maintaining these properties is easily accomplished. It should be noted, however, that while this approach overcomes a potentially long sequence of pivots each time Step 2 is performed if the point $[\lambda, v(\lambda)]$ is highly degenerate in (G) then a potentially long sequence of Steps 2 and 3 combined could occur in which a value of $\theta = 0$ is returned from Step 3. In fact, without taking appropriate measures Step 2 could encounter the same sets $S_X$ and $S_A$ encountered previously. This possibility is discussed further in Section 6.

Implementation of Step 3

Before discussing the algorithm for solving problem FPARAM it is useful to consider the simpler problem of maximizing a linear function on $P_F^k$. It is well known that optimizing a linear function on $P_F^k$ is an $NP$-complete problem. It is also well known that this problem can be solved using dynamic programming (see, for example, [15] p. 418). In the formulation of the dynamic program with $\lambda$ defining the linear function to be maximized, the recursive relation is given by

\[
\begin{align*}
\text{if } g(j-1, k) &> g(j-1, k-a_j) + \lambda_j \text{ then} \\
g(j, k) & = g(j-1, k) \\
pr(j, k) & = (j-1, k) \\
\text{else} \\
g(j, k) & = g(j-1, k-a_j) + \lambda_j \\
pr(j, k) & = (j-1, k-a_j)
\end{align*}
\]

where $g(j, k)$ conceptually represents the optimal solution to the problem

\[
\begin{align*}
\text{max} & \quad \sum_{t=1}^{j} \lambda_t x_t \\
\text{s.t.} & \quad \sum_{t=1}^{j} a_t x_t \leq k \\
& \quad 0 \leq x_t \leq 1 \\
& \quad x_t \text{ integer}
\end{align*}
\]
and the predecessor array \( pr(j, k) \) implicitly defines the \( x_j \) values by

\[
\begin{align*}
x_j &= 0 \text{ if } pr(j, k) = (j - 1, k) \\
x_j &= 1 \text{ if } pr(j, k) = (j - 1, k - a_j).
\end{align*}
\]

The value \( g(n, b_i) \) is thus the optimal solution value associated with maximizing \( \lambda x \) on \( P_F^i \).

In general, each stage \( j \) of the dynamic program can have as many as \( b_i \) states \( k \) and thus this dynamic programming formulation is only pseudopolynomial in the size of the problem it seeks to solve. However, in practice it is not necessary to consider such a large number of states. Consider a directed graph with vertex set \( \{(j, k) : j = 0, \ldots, n; k = 0, \ldots, b_i\} \) with each vertex corresponding to one possible stage/state pair in the dynamic program. Let the directed edge set correspond to the pairs defined by the recursive relation; specifically, let the edge set consist of the edges \((j, k)-(j - 1, k)\) and \((j, k)-(j - 1, k - a_j)\) for \( j = 1, \ldots, n \) and \( k = 0, \ldots, b_1 \), disallowing edges in this definition incident to nonexistent vertices. Computing \( g(n, b_i) \) can be achieved by solving the problem of finding a longest path from \((n, b_i)\) to any vertex \((0, k)\) where the edge weights are 0 for edges \((j, k)-(j - 1, k)\) and \( \lambda_j \) for edges \((j, k)-(j - 1, k - a_j)\). Clearly, only vertices and edges in this graph that can be reached by a directed path from \((n, b_i)\) need to be explicitly considered. The subgraph corresponding to these vertices and edges is easily constructed and using appropriate data structures the dynamic programming recursion can be calculated very quickly on this subgraph.

The recursive relation presented in Figure 2 is a modification to the basic dynamic programming recursion which presents a tie breaking rule in the presence of a parameterizing vector \( d \). An additional array \( h(j, k) \) is used which is conceptually defined so that \( h(j, k) = \sum_{t=1}^j d_t \cdot \overline{z}_t \) where the vector \( \overline{z} = [\overline{z}_1, \ldots, \overline{z}_j] \) is the optimal solution defined by \( pr(j, k) \). The relevance of this array is that \( g(j, k) = g(j, k) + \theta h(j, k) \) over the range of \( \theta \) for which \( \lambda + \theta d \) has the same optimal predecessor array \( pr(j, k) \) as when \( \theta = 0 \). The array \( h(j, k) \) is actually used to calculate this range on \( \theta \) as the proof of the following theorem demonstrates.

**Theorem 5** If the algorithm presented in Figure 2 is used to maximize \( \lambda x \) on \( P_F^i \) given a parameterizing vector \( d \), then \( \theta_{\text{max}} > 0 \) and for \( 0 \leq \theta \leq \theta_{\text{max}} \) the optimal predecessor array \( pr(j, k) \) remains
Given:
- a knapsack polyhedron $\mathcal{P}_F$
- a linear function $\lambda x$ to maximize on $\mathcal{P}_F$
- a direction $d$ for parametrically altering $\lambda$

Initialize:

\[
\begin{align*}
g(0, k) &= 0 \\
h(0, k) &= 0 \\
pr(0, k) &= 0 \\
\theta_{\text{max}} &= \infty
\end{align*}
\]

Recursive Relation:

\[
\begin{align*}
\text{if } g(j - 1, k) &> g(j - 1, k - a_j) + \lambda_j \text{ then} \\
g(j, k) &= g(j - 1, k) \\
h(j, k) &= h(j - 1, k) \\
pr(j, k) &= (j - 1, k) \\
\theta &= [g(j - 1, k) - g(j - 1, k - a_j) - \lambda_j]/[h(j - 1, k - a_j) + d_j - h(j - 1, k)] \\
\text{if } \theta > 0 \text{ then } \theta_{\text{max}} &= \min\{\theta_{\text{max}}, \theta\}
\end{align*}
\]

\[
\begin{align*}
\text{else if } g(j - 1, k) &< g(j - 1, k - a_j) + \lambda_j \text{ then} \\
g(j, k) &= g(j - 1, k - a_j) + \lambda_j \\
h(j, k) &= h(j - 1, k - a_j) + d_j \\
pr(j, k) &= (j - 1, k - a_j) \\
\theta &= [g(j - 1, k) - g(j - 1, k - a_j) - \lambda_j]/[h(j - 1, k - a_j) + d_j - h(j - 1, k)] \\
\text{if } \theta > 0 \text{ then } \theta_{\text{max}} &= \min\{\theta_{\text{max}}, \theta\}
\end{align*}
\]

\[
\begin{align*}
\text{else if } h(j - 1, k) &> h(j - 1, k - a_j) + d_j \text{ then} \\
g(j, k) &= g(j - 1, k) \\
h(j, k) &= h(j - 1, k) \\
pr(j, k) &= (j - 1, k)
\end{align*}
\]

\[
\begin{align*}
\text{else} \\
g(j, k) &= g(j - 1, k - a_j) + \lambda_j \\
h(j, k) &= h(j - 1, k - a_j) + d_j \\
pr(j, k) &= (j - 1, k - a_j)
\end{align*}
\]

Figure 2: Recursive Relation for Optimizing $(\lambda + \theta d)x$ on $\mathcal{P}_F^i$
optimal when $\lambda + \theta d$ is maximized on $P^t$.

Proof. Clearly, the recursive relation of Figure 2 yields an optimal predecessor array since if $g(j-1,k) \neq g(j-1,k-a_j) + \lambda_j$ the arrays $g(j,k)$ and $pr(j,k)$ are chosen in exactly the same way as in the unmodified recursive relation, and if $g(j-1,k) = g(j-1,k-a_j) + \lambda_j$ then $pr(j,k)$ can be chosen arbitrarily while yielding an optimal predecessor array.

The optimal predecessor array $pr(j,k)$ remains optimal as long as for each $(j,k)$ with $pr(j,k) = (j-1,k)$ it is true that

$$g(j-1,k) + \theta h(j-1,k) \geq g(j-1,k-a_j) + \lambda_j + \theta[h(j-1,k-a_j) + d_j]$$

and for each $(j,k)$ with $pr(j,k) = (j-1,k-a_j)$ it is true that

$$g(j-1,k) + \theta h(j-1,k) \leq g(j-1,k-a_j) + \lambda_j + \theta[h(j-1,k-a_j) + d_j].$$

The value

$$\bar{\theta} = [g(j-1,k) - g(j-1,k-a_j) - \lambda_j]/[h(j-1,k-a_j) + d_j - h(j-1,k)]$$

satisfies these expressions at equality, and it follows that in either case the largest that $\theta$ can be made without violating the appropriate inequality is $\bar{\theta}$ if $\bar{\theta} > 0$ and $\infty$ if $\bar{\theta} < 0$. When $g(j-1,k) \neq g(j-1,k-a_j) + \lambda_j$ it is not possible for $\bar{\theta}$ to be 0 so the maximum allowable increase in $\theta$ must be strictly positive in each case. When $g(j-1,k) = g(j-1,k-a_j) + \lambda_j$ the expressions reduce to

$$\theta h(j-1,k) \geq \theta[h(j-1,k-a_j) + d_j]$$

and

$$\theta h(j-1,k) \leq \theta[h(j-1,k-a_j) + d_j].$$

Alternatively stated, when $g(j-1,k) = g(j-1,k-a_j) + \lambda_j$ then the optimal predecessor array $pr(j,k)$ remains optimal for any $\theta \geq 0$ if and only if for each $(j,k)$ with $pr(j,k) = (j-1,k)$ it is true that

$$h(j-1,k) \geq h(j-1,k-a_j) + d_j$$

18
and for each \((j, k)\) with \(pr(j, k) = (j - 1, k - a_j)\) it is true that
\[
h(j - 1, k) \leq h(j - 1, k - a_j) + d_j.
\]
It can be seen that when \(g(j - 1, k) = g(j - 1, k - a_j) + \lambda_j\) the modified recursive relation of Figure 2 chooses the predecessor of \((j, k)\) to satisfy these last conditions and that \(\theta_{\text{max}} > 0\) represents the maximum allowable increase in \(\theta\) defined by pairs \((j, k)\) for which \(g(j - 1, k) \neq g(j - 1, k - a_j) + \lambda_j\). The desired result follows. 

The importance of Theorem 5 is that it can be used to show how the recursive relation presented in Figure 2 solves problem FPARAM.

**Theorem 6** The problem FPARAM associated with \(P_f\) can be solved using at most a finite number of evaluations of the recursive relation presented in Figure 2.

**Proof.** Consider the \(X\) constraint \(z \leq \lambda(x - \bar{x})\) where \(\bar{x}\) maximizes \(\lambda x\) on \(P_f\) and is determined by applying the recursive relation presented in Figure 2. Clearly, this constraint is a candidate to be in the set \(S_X\) associated with the point \([\bar{\lambda}, v(\bar{\lambda})]\) since \(v(\bar{\lambda}) = \bar{\lambda}(x - \bar{x})\). It can be determined if \([\bar{\lambda}, v(\bar{\lambda})] + \theta_{\text{max}}[d, 1]\) satisfies this constraint by determining if \(v(\bar{\lambda}) + \theta_{\text{max}} \leq (\bar{\lambda} + \theta_{\text{max}}d)(x - \bar{x}) = v(\bar{\lambda} + \theta_{\text{max}}d)\). If not, that is, if \(v(\bar{\lambda}) + \theta_{\text{max}} > v(\bar{\lambda} + \theta_{\text{max}}d)\), then since \([\bar{\lambda}, v(\bar{\lambda})] + \theta_{\text{max}}[d, 1]\) satisfies all of the \(S_X\) constraints for all \(\theta \geq 0\) by assumption it follows that the \(X\) constraint associated with \(\bar{x}\) is not in \(S_X\) and therefore the value \(\theta = 0\) together with this \(X\) constraint represent a solution to the problem FPARAM.

Thus, suppose \(v(\bar{\lambda}) + \theta_{\text{max}} \leq v(\bar{\lambda} + \theta_{\text{max}}d)\) so that \([\bar{\lambda}, v(\bar{\lambda})] + \theta[d, 1]\) satisfies all of the \(X\) constraints for \(0 \leq \theta \leq \theta_{\text{max}}\). Let \(\theta^A_{\text{max}}\) be the largest value of \(\theta \geq 0\) such that \(\bar{\lambda} + \theta d\) satisfies all of the \(A\) constraints. All of the constraints in \(S_A\) must be satisfied at equality for any \(\theta \geq 0\) by assumption and thus \(\theta^A_{\text{max}}\) can be determined by finding when \(\lambda + \theta d\) first violates each of the remaining constraints and taking the minimum of these values. If \(\theta^A_{\text{max}} \leq \theta_{\text{max}}\) then \(\theta = \theta^A_{\text{max}}\) together with the \(A\) constraint that defined \(\theta^A_{\text{max}}\) represent a solution to the problem FPARAM.
Thus, suppose that $\theta_{\text{max}}^A > \theta_{\text{max}}$ and let $\bar{\lambda} = \lambda + \theta_{\text{max}}d$. By Theorem 5 there exists a $\theta'_{\text{max}} > 0$ such that the predecessor array $pr(j, k)$ remains optimal when $\bar{\lambda} + \theta d$ is maximized on $P_F$ and so the process just described can be repeated using this new value of $\lambda$. Finite termination of the procedure is ensured by recognizing that $\theta_{\text{max}}$ is chosen so that applying the recursive relation at $\bar{\lambda}'$ corresponds to a nondegenerate parametric simplex pivot. \hfill \Box

6 Computational Results

The algorithm described in the previous section was tested by applying it to the collection of $0/1$ integer programs studied by Crowder, Johnson, and Padberg in [3]. A summary of these problems is shown in Table 1. Problems from this test set were chosen for a number of reasons. First, they represent a collection of real-world problems and thus exhibit characteristics that are not generally exhibited by randomly generated problems. Second, the problems have been solved to optimality and thus it is possible to measure how much of the gap has been closed between the optimal value of the original integer program and its linear programming relaxation. Finally, the problems are rapidly becoming a standard test set of integer programs. Crowder, Johnson, and Padberg argued that under the assumption that an integer program was sparse the polyhedron $P$ defined by the convex hull of feasible integer points often would be reasonably well approximated by $\bigcap_{i=1}^m P_F^i$. This claim was strongly supported by their computational results.

The algorithm in which the cut generation algorithm of the previous section was embedded
proceeded as follows. The linear programming relaxation of a given integer program was first solved to obtain an optimal solution \( \hat{x} \). A pass was then made through the constraints during which a Fenchel cut was sought for each of the subproblem polyhedra \( \mathcal{P}_i \). Any Fenchel cuts that were found were then appended to the original problem formulation and the process was repeated. On subsequent passes not every polyhedron \( \mathcal{P}_i \) was examined for a Fenchel cut since some of these polyhedra were clearly not defining active constraints near the optimal solution. However, in every problem except problem P2756 the algorithm did not terminate until a pass had been made in which it was demonstrated that no cutting plane existed for any polyhedron \( \mathcal{P}_i \). Thus, in these instances the algorithm provided a proof that the most recently determined optimal solution \( \hat{x} \) for the linear programming relaxation of the problem was optimal over \( \bigcap_{i=1}^m \mathcal{P}_i \). Initially it was not clear that this would be achievable within reasonable computation times since the polyhedra \( \mathcal{P}_i \) are associated with NP-complete problems. The algorithm used by Crowder, Johnson, Padberg for generating cuts, even if solved exactly rather than heuristically as is done in [3], does not provide a proof of optimality over \( \bigcap_{i=1}^m \mathcal{P}_i \).

Computational results for the algorithm described above in conjunction with the ascent algorithm for cut generation described in the previous section are shown in Tables 2 and Table 3. The column labeled \( \text{Cuts} \) is the total number of cuts appended to the problem. The columns labeled \( v_{LP}^{1,0} \), \( v_{LP}^{2,0} \), and \( v_{LP}^{3,0} \) in Table 2 give the value of the linear programming relaxation after 1, 2, and 3 minutes, respectively. The column \( v_{LP}^T \) gives the value of the linear programming relaxation after \( T \) minutes where \( T \) is given in the table. For all of the problems except P2756 the values \( v_{LP}^T \) represent the provably best gap reduction that can be achieved using only cutting planes associated with the individual knapsack constraints. This result was not achieved for problem P2756 since two of the constraints were large enough that \( v(\lambda) \) could not be maximized in reasonable computation times with the existing ascent algorithm. The values \( \Delta \text{Gap}^{1,0} \), \( \Delta \text{Gap}^{2,0} \), and \( \Delta \text{Gap}^{3,0} \) in Table 3 represent the percentage by which the gap between \( v_{LP} \) and \( v_{LP} \) was reduced in 1, 2, and 3 minutes, respectively, with \( \Delta \text{Gap}^T \) representing the percentage by which this gap was closed in \( T \) minutes.
Table 2: Cut Summary Using Ascent Algorithm

<table>
<thead>
<tr>
<th>Name</th>
<th>$v_{LP}$</th>
<th>$v_{IP}$</th>
<th>$\Delta Gap^{1.0}$</th>
<th>$\Delta Gap^{2.9}$</th>
<th>$\Delta Gap^{3.6}$</th>
<th>$\Delta Gap^{4.5}$</th>
<th>$T$</th>
<th>Cuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0033</td>
<td>2520.60</td>
<td>3089.00</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>.17</td>
<td>58</td>
</tr>
<tr>
<td>P0040</td>
<td>61796.50</td>
<td>62027.00</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>.01</td>
<td>4</td>
</tr>
<tr>
<td>P0201</td>
<td>6875.00</td>
<td>7615.00</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>.99</td>
<td>30</td>
</tr>
<tr>
<td>P0282</td>
<td>176867.50</td>
<td>258411.00</td>
<td>94.21%</td>
<td>96.31%</td>
<td>96.58%</td>
<td>98.59%</td>
<td>27.09</td>
<td>502</td>
</tr>
<tr>
<td>P0291</td>
<td>1705.10</td>
<td>5223.80</td>
<td>75.59%</td>
<td>76.96%</td>
<td>97.27%</td>
<td>99.43%</td>
<td>35.84</td>
<td>241</td>
</tr>
<tr>
<td>P0548</td>
<td>315.30</td>
<td>8691.00</td>
<td>23.05%</td>
<td>61.72%</td>
<td>74.36%</td>
<td>76.68%</td>
<td>9.65</td>
<td>364</td>
</tr>
<tr>
<td>P2756</td>
<td>2688.70</td>
<td>3124.00</td>
<td>0.56%</td>
<td>2.61%</td>
<td>3.00%</td>
<td>86.16%</td>
<td>28.68</td>
<td>487</td>
</tr>
</tbody>
</table>

Table 3: Cut Summary Using Ascent Algorithm

All computational tests were performed on a SUN SPARCserver 390.

Degeneracy proved to be a significant obstacle to the speed of the algorithm. With respect to the ascent algorithm for maximizing $v(\lambda)$, degeneracy expresses itself as a value of 0 returned by the algorithm for solving problem FPARAM. Common wisdom is that degeneracy is best dealt with simply by ignoring it. However, for the functions $v(\lambda)$ associated with the $P_f$ of the test problems this proved to be a very bad idea in some instances. Long sequences of degenerate pivots were sometimes observed that severely impeded the progress of the algorithm.

The perturbation method and Bland's anticycling rule were both considered as potential ways of overcoming the problem of degeneracy. Beyond the practical inefficiency of these two approaches they both require that the active constraints at the degenerate vertex be known explicitly and this is not the case for the problem (G). While it is possible to use these techniques while simultaneously refining the known collection of active constraints at the vertex this requires continually adding
constraints to the set $S_X$ returned from the function for solving problem FPARAM. This works against the effort to maintain a small set $S_X$ and the correspondingly small amount of work required to complete Step 2 of the algorithm.

To overcome the problem of degeneracy it was decided to choose the direction of steepest ascent in Step 2 of the algorithm rather than an arbitrary ascent direction; specifically, the direction of ascent $[d, 1]$ was chosen as the direction that maximized $1/\|d\|$. The computational expense associated with this approach, of course, is that the direction of ascent associated with each variable of positive reduced cost must be calculated, and finding each direction entails solving a square linear system in $|S_X|$ variables ($|S_X| + 1$ if the $\beta$ constraint is included in (H)). If only reduced cost information is used to choose the entering variable and the associated direction of ascent then only one linear system must be solved beyond the system solved to calculate the dual variables. In practice, while steepest ascent was much better at resolving degeneracy it was also so computationally expensive that it was only used after a sufficiently long sequence of degenerate pivots was encountered, that is, after a sufficiently long sequence of 0's was returned by the algorithm for solving problem FPARAM. This strategy of using a degeneracy mechanism only when needed proved to be the best strategy by far.

The set $\bar{A}$ as defined in Section 4 was chosen as the domain over which to maximize $v(\lambda)$ with $\beta$ chosen to be 0.5 or $n + 1$. The arguments presented in Section 4 suggested that the domain associated with $\beta = 0.5$ would tend to generate Fenchel cuts with good polyhedral characteristics. In practice, the difficulty with this domain is that the easily generated extreme points of (G) are highly degenerate and thus slow the speed of the algorithm considerably. To see this, note that the most easily generated extreme points of (G) correspond to $\lambda_k = 0$ for all $k$ with $z = v(\lambda)$ or to $\lambda_j = 0.5$ for some $j$, $\lambda_k = 0$ for $k \neq j$, and $z = v(\lambda)$. The case $\lambda_k = 0$ is as degenerate as possible since every $x^i \in E(P^i_F)$ maximizes $0x$ on $P^i_F$ and thus $v(\lambda) = \lambda \hat{x} - \lambda x^i$ for every $x^i \in E(P^i_F)$. The case with $\lambda_j = 0.5$ for some index $j$ is almost as bad since with only one positive $\lambda_j$ the number of $x^i \in E(P^i_F)$ maximizing $\lambda x$ on $P^i_F$ remains very large.
With $\beta = n + 1$ the domain $\overline{A}$ has the feasible extreme point $\lambda_k = 1$ for all $k$. In general, degeneracy associated with this value of $\lambda$ is small or nonexistent since the expected number of $x^i \in E(P^k_{F})$ maximizing $\lambda x$ on $P^k_{F}$ is small. In practice, maximization of $v(\lambda)$ was up to ten times faster on the domain $\overline{A}$ defined by $\beta = n + 1$ than $\beta = 0.5$ because of the type of degeneracy just described. Nonetheless, the domain defined by $\beta = 0.5$ was not discarded since in some problems the strength of the cuts generated by this domain were sufficiently strong to warrant the additional time required to generate them. At any given invocation of the cut generation algorithm the domain was chosen based upon the past history of cuts generated for the particular subproblem, with both the estimated strength of the cut and the estimated time to generate the cut taken into consideration. This proved to be the best strategy since it did not allow either domain to be ruled out entirely.

In order to place the computational results in context, an algorithm identical to the algorithm described above was implemented except that the cuts were generated using generalized programming to maximize $(G)$. Specifically, the generalized programming algorithm proceeded as follows. The problem $(G)$ was approximated using all of the $A$ constraints and some subset of the $X$ constraints; initially, one arbitrarily chosen $X$ constraint was used. The resultant linear programming approximation to $(G)$ was then solved to obtain a vector $[\overline{A}, z^*]$. The linear function $\overline{A}x$ was then maximized on $P^k_{F}$ to determine $v(\overline{A})$. If $v(\overline{A}) = z^*$ it follows that $[\overline{A}, z^*]$ is feasible and optimal for a relaxation of $(G)$ and must therefore be optimal for $(G)$. In this case the algorithm terminates with the Fenchel cut $\overline{A}x \leq v(\overline{A})$. If $v(\overline{A}) < z^*$ the $X$ constraint associated with the $x^i \in E(P^k_{F})$ satisfying $\overline{A}x^i = v(\overline{A})$ is violated by $[\overline{A}, v(\overline{A})]$. In this case, the new $X$ constraint associated with $x^i$ is included in the approximation of $(G)$ and a new optimal solution to this approximation is sought.

A number of modifications were made to the basic generalized programming algorithm in an effort to improve its performance. A simpler version of the dynamic programming algorithm for maximizing $v(\lambda)$ was required for the generalized programming algorithm than for the ascent algorithm since generalized programming requires only $v(\lambda)$ and no parametric information. Constraints were selectively dropped from the approximation to $(G)$ when they had not been active at the optimal
One of the characteristics of generalized programming is that a sequence of linear programs must be solved in which an optimal solution is known for one linear program and the next problem in the sequence is formed by adding a single constraint violated by this solution. Such a sequence is a classic example of a collection of linear programs to which the dual simplex algorithm can be applied to improve the speed with which each problem can be solved. An unreleased version of the CPLEX callable library that makes use of the dual simplex algorithm was used to solve this sequence of linear programs.

With the modifications just described the generalized programming algorithm was extremely efficient. Significant time was spent developing this algorithm before the author felt the potential...
of the generalized programming approach had been reached. In fact, it was the limitations of this approach that led the author to develop the ascent algorithm described in this paper. Computational times paralleling those presented in Tables 2 and 3 are presented in Tables 4 and 5. As can be seen, the ascent algorithm significantly outperformed generalized programming. To fully appreciate these comparative results it is important to emphasize again that the generalized programming algorithm represented an extremely efficient implementation using a state-of-the-art linear programming code. The difference in running times between the two algorithms thus represents a fundamental difference in the algorithms themselves and not their implementations. In fact, there remain areas of potential improvement for the ascent algorithm that have not been investigated.

Initially a major effort was made to develop an ascent algorithm based on subgradients but subgradient techniques were abandoned when they proved hopelessly inadequate. Even when a reasonable sequence of iterates $\lambda^i$ could be generated — and this in itself was not always easy to accomplish — the empirical rate of convergence was generally so poor that the algorithm was useless for all practical purposes. Subgradient algorithms are so intuitive, simple to code, and generally well-regarded in the literature that it took the author a long time to realize how inadequate they can be. However, it was a full appreciation of the information that subgradient techniques disregard — namely, an intelligent choice of ascent direction — that led to the algorithm presented in this paper.

7 Conclusions

An ascent algorithm for solving the dual maximization problem arising in the generation of Fenchel cuts has been presented and described in detail for the dual problem associated with knapsack polyhedra. An implementation of this algorithm was then applied to generate cutting planes for a collection of integer programs and to provably optimize over the intersection of the knapsack polyhedra defined by the constraints of these problems. It was also demonstrated that the ascent algorithm was faster than generalized programming and preferable to subgradient techniques.
A number of open questions remain to be addressed. A significant obstacle to the speed of
the ascent algorithm is degeneracy. One method of avoiding degeneracy is to avoid extreme points
of the polyhedron defined by (G). While extreme points cannot and really should not be avoided
altogether, restricting ascent directions to edges of the polyhedron (G) may tend to cause a sequence
of degenerate pivots to last longer than is necessary. For example, a call to function FPARAM may
discover a direction of ascent long before a direction of ascent that is an edge of (G) is found.
For many classes of linear programs the work required to find a new basis is likely to outweigh the
additional work required to find a direction of ascent corresponding to an edge. However, degeneracy
is so common and the dimension of the problems encountered is so small that for the dual functions
associated with knapsack polyhedra it may be computationally advantageous to allow the ascent
algorithm to stray from extreme points of (G).

In general, the algorithm for solving problem FPARAM described in Section 5 must be solved in
its entirety for each new direction [d, 1] even if the feasible solution [X, v(X)] for (G) does not change.
However, if [d, 1] does not change, even if [X, v(X)] changes, then it is possible to resolve FPARAM
using information available from the previous solution of this problem. Thus, for example, an ascent
direction [d, 1] might be chosen and [X, v(X)] modified until [d, 1] ceased to be an ascent direction
(intermediate values of [X, v(X)] at which FPARAM would need to be resolved would correspond to
points where the predecessor list in the algorithm for solving FPARAM changed). In conjunction
with allowing the ascent algorithm to stray from extreme points of (G), using previous solutions of
problem FPARAM to resolve this problem could prove to be very efficient computationally.

The ascent algorithm presented in this paper achieved good performance by taking advantage of
the underlying combinatorial structure of the subproblem defining the dual maximization problem
rather than disregarding this fundamental information as alternative techniques do. It is more than
conceivable that for some subproblems there exist efficient combinatorial methods for choosing the
ascent direction in Step 2 of the ascent algorithm or even efficient combinatorial methods for solving
the dual maximization problem itself. The types of combinatorial problems that arise in this context
deserve further attention.

Finally, it is worth noting that while the ascent algorithm was presented as it related to Fenchel cuts, this algorithm can be applied to dual problems that arise in the context of solving Lagrangian problems as well. As with Fenchel cuts, all that is necessary to apply the ascent algorithm is an oracle for solving problem FPARAM.
References


30