

Fully Polynomial Approximation in Multi-Criteria Combinatorial Optimization

Hershel M. Safer[†], James B. Orlin[‡], and Moshe Dror^{‡*}

[†] Weizmann Institute of Science

[‡] Massachusetts Institute of Technology

* University of Arizona

February 22, 2004

DRAFT

Keywords: combinatorial optimization, NP-hard, polynomial approximation, multi-criteria optimization

Abstract

In this paper, we extend the notion of a fully polynomial time approximation scheme (FPTAS) to multi-criteria optimization problems. Necessary and sufficient conditions for the existence of an FPTAS are presented. We also develop a new form of reduction to prove the existence of an FPTAS for a variety of multi-criteria problems. The generic problem used to derive most of the results is the Source-to-Tree (STT) Network Flow problem. We establish the existence of an FPTAS for several multi-criteria knapsack, scheduling, and production problems.

1 Introduction

Suppose that we are given a graph $G = (V, E)$ and the task of finding a path between two specified nodes in V that is both shortest and cheapest. Finding the shortest path is easy, as is finding the cheapest path; but finding a path that is both shortest and cheapest is not.

In general, a graph will not have a single path that is both shortest and cheapest. Instead, a solution to this problem consists of a *set* of paths that are *efficient* or *Pareto optimal*. Each of these paths is optimal in the sense that any other path which is shorter must be more expensive, and any other path which is cheaper must be longer. No path can be as short as and cheaper than, or as cheap as and shorter than, an efficient path.

For some multi-criteria combinatorial optimization problems, just establishing if a feasible solution is efficient is NP-hard. In the example above, determining if a specific path is efficient with respect to distance and cost is NP-complete (Garey and Johnson, 1979). In contrast to this, in multi-criteria linear programming, determining if a given feasible solution is efficient is easy: it can be done by solving a single additional linear program (see, e.g., Ecker and Kouada, 1975).

Furthermore, for many multi-criteria problems, asking for the set of efficient solutions is not even in NP, because the size of the solution set might not be bounded by any polynomial in the size of the problem's encoding. In some cases, the cardinality of the efficient set is exponential or even uncountable.

In view of these difficulties, we propose finding *approximate* solutions to multi-criteria combinatorial optimization problems. For any $\epsilon > 0$, an ϵ -approximate solution must satisfy two conditions in addition to feasibility. First, an approximate efficient set must be small (i.e., of size polynomially bounded in the problem's encoding and in $1/\epsilon$). Second, for any efficient solution, the approximately-efficient set must include a point whose value is no more than a fraction $(1 + \epsilon)$ larger in each component (assuming minimization) than the criteria vector for the specified efficient solution.

Methods for finding approximate solutions to single-criterion optimization problems have been discussed extensively. In the single-criterion context, an ϵ -approximation algorithm finds a solution

whose relative error from optimality is no greater than ϵ . For some problems, such algorithms exist only for values of ϵ that are at least as large as some positive lower bound, unless $P = NP$. For some other problems, ϵ -approximate algorithms exist for any positive value of ϵ . In the latter case, two classes of algorithms have received primary attention. For any fixed $\epsilon > 0$, a *polynomial time approximation scheme* (PTAS) runs in time polynomial in the size of the instance, and a *fully polynomial time approximation scheme* (FPTAS) runs in time polynomial in the size of the instance and in $1/\epsilon$.

Approximation in the multi-criteria context has received rather less attention. Approximately efficient solutions are, however, known to exist. Papadimitriou and Yannakakis (2000) proved that for any multi-criteria optimization problem with a fixed number of criteria, and any $\epsilon > 0$, there exists a set of ϵ -approximate solutions of cardinality polynomial in the size of the problem's input and $1/\epsilon$. However, this existence result does not guarantee the possibility of polynomial-time construction.

The present paper addresses the existence of fast algorithms for finding approximate solution sets. It presents conditions under which an ϵ -efficient set for a *multi-criteria* problem can be computed in polynomial time and space in both the size of the problem's encoding and $1/\epsilon$, that is, it identifies necessary and sufficient conditions for the existence of an FPTAS for a multi-criteria discrete optimization problem.

The necessary and sufficient conditions are developed in the context of general combinatorial optimization. The paper bridges the gap between the necessary and sufficient conditions by identifying a smaller subclass of problems that is appropriate for both; the results are therefore of interest even in the single-criterion case. The appendix introduces a new form of reduction that is used to simplify application of the main theorem.

The remainder of the paper discusses FPTASs for a variety of problems. The multi-criteria Source-To-Tree Network Flow (STT) problem is introduced and an FPTAS is described for generating an ϵ -efficient solution for variants of this problem. This results in establishing the existence of FPTASs for multi-criteria flow, knapsack, and scheduling problems. We conclude by establishing the existence of an FPTAS for the single-commodity dynamic lot sizing problem.

An ϵ -efficient set may contain a large number of solutions, even though it has cardinality polynomial in the size of the problem's encoding. Given an approximate solution set, a decisionmaker will be faced with the question of how to select one or several solutions. This choice implies an explicit or implicit tradeoff between the criteria. The issue of choosing between alternative approximately-efficient solutions is not addressed in this paper.

In the multi-criteria literature, a popular approach for selecting a solution from a set of efficient solutions is by minimizing some measure of distance (a norm) between a solution and the "ideal" r -vector of optimal single objective values (see Steuer, 1986). Unfortunately, this approach cannot be extended to the case of ϵ -efficiency because the notion of " ϵ -ideal" cannot be modified to represent an r -vector of values, each component being $(1 + \epsilon)$ relative distance from its single-objective optimum.

This paper is based on and borrows heavily from Safer and Orlin (1995a, 1995b). Readers interested in details of the historical perspective and the related mathematical foundations should consult these two working papers. We also note two important books on single-criterion approximation algorithms which appeared in 1997 and 2001, by Hochbaum and Vazirani respectively. These books contain all of the basic and most of the advanced results in that area.

2 Basic Terminology, Definitions, and Notation

The problem of finding a path between two given nodes s and t in G that is both cheap and short serves to illustrate and clarify concepts and definitions. A very fine analysis and discussion of this specific problem has been presented by Warburton (1987). We have already noted that this problem is easy when using either criterion by itself but is NP -hard with both criteria simultaneously. The computational and conceptual complications that arise from adding criteria to easy single-criterion problems motivate our investigation of approximation algorithms for multi-criteria optimization.

We begin with basic approximation concepts for single-criterion problems. An algorithm for a problem runs in *pseudo-polynomial time* if it solves any instance of the problem in time polynomial in the size of the instance (that is, the space required by a compact binary encoding of the instance)

and in the value of the largest integer that appears in the instance description.

A necessary condition for the existence of an FPTAS for a single-objective problem is the existence of a pseudo-polynomial time algorithm for the problem (Garey and Johnson, 1978). This is, however, not sufficient (if $P \neq NP$): the two-dimensional binary knapsack problem can be solved by a pseudo-polynomial algorithm but does not have an FPTAS (Gens and Levner, 1979). The existence of an FPTAS for an NP -hard problem is the theoretically most promising result about solvability if $P \neq NP$, so identifying ever larger classes of NP -hard problems for which the existence of an FPTAS can be guaranteed is of considerable interest.

Papadimitriou and Steiglitz (1982) and Orlin (1982) have identified subclasses of pseudo-polynomial time algorithms, each of which implies the existence of an FPTAS for the corresponding NP -hard problem. However, some problem-specific FPTAS existence results are outside these classes (e.g., van Hoesel and Wagelmans, 2001). An interesting class of NP -hard problems with FPTASs has also been described by Woeginger (2000) in terms of the characteristics of the corresponding dynamic programming formulation that Woeginger calls “DP-benevolent.” This paper extends the sufficiency conditions for existence of FPTASs both for single-objective problems and to the more general setting of multi-criteria problems. The discussion focuses on the latter, but notes the implications for the former. The paper is mostly self-contained, but basic familiarity with Garey and Johnson (1979) would be helpful.

The next two subsections introduce definitions relating to problem statements, problem solutions, and solution algorithms.

2.1 Problem Statements

This section describes the notation used to represent problems and instances and the assumptions that we make about them. The following conventions are observed, except where specified. Logarithms are to the base 2. Vector norms are infinity norms, that is $\|x\| = \max\{|x_j| : x = (x_1, \dots, x_k)\}$. The vector e_k is a k -dimensional vector of 1's. The set $Z^{n \geq}$ is the n -dimensional nonnegative integer grid and $Z^{n+} = Z^{n \geq} \setminus \{(0, \dots, 0)\}$ is the positive integer grid.

For simplicity of exposition, we assume, without loss of generality, minimization in each criterion, except where noted otherwise. Safer and Orlin (1995a) address maximization and combinations of minimization and maximization in different components of an objective function.

An n -dimensional *feasible set* is a bounded set $S \subseteq Z^{n \geq}$. Since S is bounded, there is an integer $u < \infty$ such that for any $x \in S, \|x\| \leq u$. The smallest integer u with this property is defined as the norm of S , i.e., $\|u(S)\| \equiv u(S) = \min_u \{\|x\| \leq u : x \in S\}$.

An r -criteria objective function f maps a feasible set S into Z^{r+} ; that is, for any $x \in S$, $f(x)$ is an r -dimensional positive integer vector. We assume that for any $x \in S$, any objective function component f_i , and any finite $M_i \in Z^+$, the truth of the statement “ $f_i(x) \leq M_i$ ” can be verified in time polynomial in the encoding of x and M_i , $i = 1, \dots, r$, or symbolically, in time $O([n \times \log(\|x\|) \times \log(M_i)]^k)$, for some integer $k < \infty$. In addition, we assume that $f(x)$ can be evaluated in time polynomial in the encoding of x and $f(x)$ for any $x \in S$, that is, in time $O([n \times r \times \log(\|x\|) \times \log(\|f(x)\|)]^k)$, for some integer $k < \infty$. To avoid the issue of intractability because of the number of criteria r , throughout this paper we assume that r is not a function of S , but is rather fixed for each family of problems.

An instance of an optimization problem is represented by a pair $I = (S, f)$, where for positive integers n and r , S is an n -dimensional feasible region and f is an r -criteria objective function. An optimization problem $\Pi = (\Xi, \mathcal{F}, Opt)$ is a collection of instances (S, f) with $S \in \Xi$ and $f \in \mathcal{F}$.

An instance of a feasibility problem is represented by a 3-tuple $I = (S, f, M)$. The first two components are the same as for the optimization instance. The final component, M , is the vector of target values that we would like to achieve (see the next subsection for precise details). A feasibility problem $\Pi = (\Xi, \mathcal{F}, Feas)$ is a collection of instances (S, f, M) with $S \in \Xi$, $f \in \mathcal{F}$, and $M \in Z^{r+}$.

The largest value of an optimization instance $I = (S, f)$ is $M_v(I) = \max\{\|f(x)\| : 0 \leq x \leq u(S)\}$. For a feasibility instance $I = (S, f, M)$ we are not concerned with values of $f_i(x)$ that are larger than M_i , so the largest value of I is $M_v(I) = \|M\|$.

The length $L(I)$ of an instance $I = (S, f)$ or $I = (S, f, M)$ is the number of bits needed to encode both S and $M_v(I)$.

The value of $M_v(I)$ for an optimization instance I can be difficult to determine exactly, but a close approximation (i.e., within a factor that is polynomial in $L(I)$) can generally be found fairly quickly. We assume that for any instance I , a close finite upper bound $\bar{M}_v(I)$ of $M_v(I)$ can be found in time polynomial in $L(I)$. That is, for some $k_1, k_2 \in \mathbb{Z}^+$, time $O([L(I)]^{k_1})$ is sufficient to find a bound $\bar{M}_v(I)$ satisfying $M_v(I) \leq \bar{M}_v(I) \in O(M_v(I) \times [L(I)]^{k_2})$.

2.2 Solutions and Algorithms

We now present the notation used to describe problem solutions and algorithms for finding them. A solution to an optimization instance $I = (S, f)$ is a minimal-cardinality set of feasible points whose f values constitute the efficient frontier. We refer to efficient sets in decision space and to the efficient frontier in criteria space.

A solution to a feasibility instance $I = (S, f, M)$ is a point x with $f(x) \leq M$, if such a point exists; that is, a point whose objective function values achieve the specified target values. Note that a statement that such an x exists is not sufficient; the solution consists of a suitable value of x .

An algorithm for an optimization or a feasibility problem Π is *V-pseudo-polynomial* (VPP), if for any instance $I \in \Pi$, the algorithm finds a solution for I in time polynomial in the length and the largest value of the instance, i.e., in time $O([L(I) \times M_v(I)]^k)$, for some $k \in \mathbb{Z}^+$. If a VPP algorithm exists for Π , then Π is said to “have a VPP algorithm.”

The “V” stands for “value”; so a VPP algorithm is essentially a pseudo-polynomial algorithm for which the integer from the problem statement that appears in the time bound is from the objective function value. A pseudo-polynomial algorithm for which an integer from the description of the feasible region appears in the time bound might not be VPP.

For an r -vector v and any $\epsilon > 0$, the set of points that are within a factor ϵ of v in each component is called an ϵ -ball around v . The balls are actually rectangles rather than spheres because the difference from v is measured separately for each dimension. We note that the balls grow bigger as the values of v increase because ϵ is a relative measure.

An ϵ -efficient solution to an optimization instance is a minimal-cardinality set of feasible points with the property that each point on the efficient frontier is close to the value of at least one point of the ϵ -efficient solution set, that is, the former point is inside a closed ϵ -ball around the latter point. This means that if we draw an ϵ -ball around the values of all the ϵ -efficient points, the entire efficient frontier will be covered. Formally, an ϵ -efficient solution for an instance $I = (S, f)$ is a minimal set $Y \subseteq S$ such that for each efficient $x \in S$, there is a point $y(x) \in Y$ such that $f_i(y) \leq (1 + \epsilon)f_i(x)$, $i = 1, \dots, k$. Note that for ϵ -efficiency, a strict “ $<$ ” on at least one coordinate is not required, though it is required for efficiency.

An algorithm for an optimization problem $\Pi = (\Xi, \mathcal{F}, Opt)$ is a fully polynomial time approximation scheme (FPTAS) if for any $\epsilon > 0$ and instance $I \in \Pi$, the algorithm finds an ϵ -efficient solution for I in time polynomial in the length of the instance and in $1/\epsilon$, that is, in time $O([L(I)/\epsilon]^k)$, for some $k \in \mathbb{Z}^+$. If some FPTAS exists for Π , then Π is said to “have an FPTAS.”

The notion of ϵ -efficiency is defined only for optimization problems, not for feasibility problems.

3 VPP Algorithms and FPTASs

We start this section by showing that the existence of a VPP algorithm for an optimization problem Π is a necessary condition for the existence of an FPTAS for Π . We then show that the converse is not true and define some relevant regularity conditions on problems. The section closes with the paper’s key theoretical result, that under the regularity condition, the existence of a VPP algorithm is sufficient to guarantee the existence of an FPTAS.

Theorem 1: Consider an optimization problem $\Pi = (\Xi, \mathcal{F}, Opt)$. If Π has an FPTAS, then Π has a VPP algorithm.

Proof: Suppose that an FPTAS \mathcal{H} for Π is specified. We construct a VPP algorithm for Π as follows. Let $I = (S, f)$ be an n -variable instance of Π .

First, determine the value $\bar{M}_v(I)$ and set $\epsilon \leftarrow 1/(1 + \bar{M}_v(I))$. Then solve I using \mathcal{H} with accuracy $(1 + \epsilon)$.

To prove that this construction constitutes a VPP algorithm for Π we have to show that the algorithm runs in VPP time and finds the solution to Π .

We assumed that there exists a $k_1 \in Z^+$ such that the close upper bound $\bar{M}_v(I)$ of $M_v(I)$ can be determined in $O([L(I)]^{k_1})$ time. Since \mathcal{H} is an FPTAS for Π , it solves Π in time $O([L(I)/\epsilon]^{k_2}) \subseteq O([L(I) \times \bar{M}_v(I)]^{k_3})$, for some $k_2, k_3 \in Z^+$. Therefore the algorithm \mathcal{H} takes VPP time.

Suppose that the algorithm returns a set $Y \subseteq S$; then for any efficient point $x \in S$, there is some $y(x) \in Y$ such that $f(y) \leq (1 + \epsilon)f(x)$. Since $f_i(x) \leq M_v(I) \leq \bar{M}_v(I)$, $i = 1, \dots, r$, it follows that $\epsilon f_i(x) \leq \bar{M}_v(I)/(1 + \bar{M}_v(I)) < 1$, $i = 1, \dots, r$. But each f_i maps into Z^+ , implying that “fractions do not count,” and so $f(y) \leq f(x)$. But x is Pareto optimal, and therefore $f(y) = f(x)$. So Y is an exact solution of Π . \square

The converse of Theorem 1 cannot be true unless $P = NP$. We demonstrate this with a problem that has a VPP algorithm but no FPTAS, unless $P = NP$.

Example 1: Minimum Cost Exact Cover by 3-Sets (MIN-X3C)

Instance: (n, m, A, B, c) and $K \in Z^+$. The number of triples n and the solution size m are non-negative integers with $m \leq n$; $A = \{a_1, \dots, a_{3m}\}$ is the set of items; and B is a collection of n triples $B_j \subseteq A$ with $|B_j| = 3$ and cost $c_j \in Z^{\geq}$, $j = 1, \dots, n$. An exact cover of A is a set $B' \subseteq B$ with $|B'| = m$ such that $\cup_{B_j \in B'} B_j = A$.

Question: Is there an exact cover \mathcal{B} of A such that $\sum_{j: B_j \in \mathcal{B}} c_j \leq K$?

Setting each $c_j = 1$ yields the *Exact Cover by 3-Sets (X3C)* problem and the cost question asks about the cardinality of an exact cover. X3C is known to be strongly NP -complete and so is MIN-X3C (Garey and Johnson, 1979). Therefore no FPTAS exists for X3C unless $P = NP$.

Now consider a restriction of MIN-X3C in which each $c_j \in [2^{n-1}, 2^n]$, $j = 1, \dots, n$. An instance I of the restricted MIN-X3C has an encoding of length $L(I) \in \Theta(n \log(m) + n \log(c_{max})) \subseteq \Omega(n^2)$ and largest value $M_v(I) \in \Omega(c_{min}) \subseteq \Omega(2^n)$. Since MIN-X3C can be solved by enumeration in time $O(n2^n) \subseteq O(L(I) \times M_v(I))$, it has a VPP algorithm. It cannot, however, have an FPTAS unless $P = NP$. Therefore the converse of Theorem 1 is not true unless $P = NP$. \square

3.1 Regularity Conditions

One way to create an FPTAS is to use a VPP algorithm on a version of the original instance in which some precision is lost in the objective function by dropping the lower-order bits in the objective function values. This approach is called *scaling*. Although solutions to the scaled instances may not exactly solve the original instance, the resulting loss of accuracy may be acceptable because an FPTAS need only generate approximate solutions. The key is to find a scaling process that reduces the running time sufficiently without losing too much accuracy. To circumvent the difficulties uncovered when using the scaling approach for FPTAS construction, we require that a problem meet the regularity conditions discussed in this section.

Definition: (Objective Function Scaling) Let f be an r -criteria function and $t \in Z^{r+}$. The function f scaled by t , written $g = \lfloor f/t \rfloor$, is defined by $g_i(x) = \lfloor f_i(x)/t_i \rfloor$, $i = 1, \dots, r$.

The following inequalities highlight some properties of scaling which are used later. If $v_i \geq 0$ and $t_i > 0$, then

$$v_i - t_i < t_i \lfloor v_i/t_i \rfloor \leq v_i < t_i \lfloor v_i/t_i \rfloor + t_i \tag{1}$$

The idea behind scaling is that one first scales and then applies a VPP algorithm. Unfortunately, an algorithm that is VPP using the original objective function need not be VPP using the scaled objective function. Just consider an instance of the restricted version of MIN-X3C scaled by $t = 2^n$, which creates an instance of X3C. Although the former has a VPP algorithm, the scaled instance does not unless $P = NP$. To avoid this problem with scaling, the following regularity condition is imposed.

Definition: (Closure under scaling) A family \mathcal{F} of r -criteria functions is said to be *closed under scaling* if $\lfloor f/t \rfloor \in \mathcal{F}$ for any $f \in \mathcal{F}$ and any $t \in Z^{r+}$.

Accuracy. Consider what happens when a VPP algorithm is used to solve an instance with a scaled cost function. Using scale factors that are powers of two is convenient because the loss of precision can be measured as a number of bits. For some $p \in Z^{\geq}$, set $t_i = 2^{p_i}$, $i = 1, \dots, r$, and

let $g = \lfloor f/t \rfloor$ be a scaled objective function. Let C_t be the (exact) solution of the scaled instance $I_t = (S, g \equiv \lfloor f/t \rfloor)$. For a given value $f(x)$, we now determine an appropriate value of the vector p so that C_t gives an ϵ -approximation of $f(x)$. Afterward we will see how to extend this approach to all values of $f(x)$.

By definition, C_t contains a point $y(x)$ with $g(y) \leq g(x)$. In order for $f(y)$ to be an ϵ -approximation of $f(x)$, p must be chosen so that $f_i(y) \leq (1 + \epsilon)f_i(x)$, $i = 1, \dots, r$. Since $g_i(y) \leq g_i(x)$, multiplying through by t_i and applying Equation 1 shows that $f_i(y) \leq f_i(x) + t_i$. So p_i must be chosen so that $t_i \leq \epsilon f_i(x)$. When $\epsilon f_i(x) \geq 1$, setting

$$p_i = \lfloor \log(\epsilon f_i(x)) \rfloor \tag{2}$$

will guarantee that $f_i(y)$ is close enough to $f_i(x)$. When $\epsilon f_i(x) < 1$, setting $p_i = 0$ and using the original objective function will work. Setting the value of the scaling vector t in this way ensures that the solution to the scaled instance will be sufficiently accurate, at least for this particular value of $f(x)$.

Running time. An FPTAS would have to solve an appropriate scaled instance I_t for every value $f(x)$, or at least for those on the efficient frontier. Rather than consider each value of $f(x)$ separately, we can address them all by solving an instance I_t for each scaling vector t with components that are powers of 2 up to $\epsilon \bar{M}_v(I)$. Specifically, we will solve I_t for each vector t with $t_i = 2^{2^{p_i}}$ for $p_i = 0, \dots, p^*$, where $p^* = \max\{\lfloor \log(\epsilon \bar{M}_v(I)) \rfloor, 0\}$, $i = 1, \dots, r$.

So $\eta = (1 + p^*)^r$ is the number of instances that will be solved. Let τ the maximum time needed to solve a single instance. If $\epsilon \bar{M}_v(I) < 1$, then $\eta = 1$. Otherwise,

$$\eta \leq (1 + \log(\epsilon \bar{M}_v(I)))^r \leq (1 + \log(\epsilon) + \log(\bar{M}_v(I)))^r .$$

Note that as the required accuracy decreases (i.e., ϵ increases), the number of instances increases.

In order to bound the number of solved instances, if $\epsilon > 2$, it will be replaced by $\epsilon' = 2$; no

accuracy is lost if a tighter error bound is used. With this modification to the solution scheme,

$$\eta \leq (2 + \log(\bar{M}_v(I)))^r \in O([L(I)]^{k_1})$$

for some $k_1 \in Z^+$, because $\log(\bar{M}_v(I)) \leq L(I)$ and r is fixed.

Since the algorithm used to solve the instance I_t is VPP for the scaled problem, the maximum time needed to solve a single instance is $\tau \in O([L(I_t) \times M_v(I_t)]^{k_2})$, for some $k_2 \in Z^+$. If $M_v(I_t)$ is large, then τ will be large as well. However, instances with large values of $M_v(I_t)$ are superfluous. For any $x \in S$, define $t_i = 2^{p_i}, i = 1, \dots, r$, as before. Then, keeping $\epsilon' \leq 2$,

$$g_i(x) = \lfloor f_i(x)/t_i \rfloor \leq f_i(x)/t_i = f_i(x)/2^{\lfloor \log(\epsilon' f_i(x)) \rfloor} < 2/\epsilon' < \lfloor 4/\epsilon' \rfloor.$$

This means that scaling to attain a relative accuracy of ϵ allows calculations to be restricted to objective function values that do not exceed $\lfloor 4/\epsilon \rfloor$. Restricting consideration to instances I_t in which $M_v(I_t) \leq \lfloor 4/\epsilon' \rfloor = \mu^*$ is therefore sufficient.

Definition: The limits on the values of the objective function values are called *box constraints*. That is, the function f with box constraints M , written $g = \min\{f, M\}$, is defined by $g_i(x) = \min\{f_i(x), M_i\}, i = 1, \dots, r$.

Since we assumed that the truth of statements like “ $f_i(x) \leq M_i$ ” can be determined in time polynomial in the size of x and M_i , box-constrained functions can be computed efficiently.

An algorithm that is VPP for a problem with box constraints requires less running time than or the same time as it does for the unconstrained problem, depending if the unconstrained problem has function values larger than the bound or not. The box constraints are used to avoid spending too much time solving any particular instance.

Definition: A family \mathcal{F} of r -criteria functions is said to be *closed under box constraints* if for any $f \in \mathcal{F}$ and any $M \in Z^{r+}$, $\min\{f, M\} \in \mathcal{F}$.

When we restrict the family \mathcal{F} of r -criteria functions to be closed under scaling, and further require \mathcal{F} to be closed under box constraints, the time τ required to solve a particular instance can

be no greater than $O([L(I_t)/\epsilon']^{k_3})$, for some $k_3 \in \mathbb{Z}^+$. The total time required to solve all the scaled instances is therefore bounded by $\eta \times \tau \in O([L(I)/\epsilon']^k)$, for some $k \in \mathbb{Z}^+$. Since $\epsilon' = \epsilon$ or $\epsilon' = 2$, the FPTAS running time requirements are satisfied.

Still, in order to extend the class of problems for which the existence of an FPTAS is guaranteed, we need to examine a weaker set of regularity restrictions. In particular, the class of additively separable functions is not closed under scaling or box constraints. The next two definitions, of quasi-closure under scaling and under box constraints, are just the regularity conditions appropriate for the class of additively separable functions.

Definition: Let \mathcal{F} be a family of r -criteria functions. Suppose that for any $f \in \mathcal{F}$ and any $t \in \mathbb{Z}^{r+}$, there is a $g \in \mathcal{F}$ such that for any $x \in \mathbb{Z}^{n \geq}$:

$$\text{if } t_i = 1, \text{ then } g_i(x) = f_i(x);$$

$$\text{if } t_i > 1, \text{ then } \lfloor f_i(x)/t_i \rfloor - n + 1 \leq g_i(x) \leq \lfloor f_i(x)/t_i \rfloor;$$

for $i = 1, \dots, r$. Then the family \mathcal{F} is said to be *quasi-closed under scaling* and the function g is called a *scaling neighbor* of f with respect to t .

Definition: Let \mathcal{F} be a family of r -criteria functions. Suppose that for any $f \in \mathcal{F}$ and any $M \in \mathbb{Z}^{r+}$, there is a $g \in \mathcal{F}$ such that for any $x \in \mathbb{Z}^{n \geq}$:

$$\text{if } f_i(x) \leq M_i, \text{ then } g_i(x) = f_i(x);$$

$$\text{if } f_i(x) > M_i, \text{ then } M_i \leq g_i(x) \leq \min\{f_i(x), nM_i\};$$

for $i = 1, \dots, r$. Then the family \mathcal{F} is said to be *quasi-closed under box constraints* and the function g is called a *box constraint neighbor* of f with respect to M .

Clearly the class of functions closed under scaling is also quasi-closed under scaling and the class of functions closed under box-constraints is also quasi-closed under box-constraints, but the reverse is not true.

3.2 FPTAS Sufficient Conditions

We now prove that the quasi-closure conditions and a VPP algorithm are sufficient for the existence of an FPTAS for a problem with separable objective functions. The method is a quite general construction based on the VPP algorithm. The FPTAS so derived may not be the most efficient

approximation scheme possible; it can typically be improved by adapting it to the special characteristics of the problem. This result is primarily useful if a scaling neighbor and box constraint neighbor of a function are reasonably easy to find for each $f \in \mathcal{F}$.

Theorem 2: Consider an optimization problem $\Pi = (\Xi, \mathcal{F}, Opt)$, where \mathcal{F} is a class of r -criteria functions that is quasi-closed under both scaling and box constraints. If Π has a VPP algorithm, then Π has an FPTAS.

Proof: Suppose that a VPP algorithm \mathcal{A} for Π is specified. Let $I = (S, F)$ be an n -variable instance of Π and $\epsilon > 0$. Define

$$\begin{aligned} \epsilon' &= \frac{1}{n} \min\{\epsilon, 2\}, \\ \mu &= \left\lfloor \frac{4}{\epsilon'} \right\rfloor e_r, \\ p^* &= \lfloor \log(\epsilon' \bar{M}_v(I)) \rfloor \text{ if } \epsilon' \bar{M}_v(I) \geq 1, \text{ and } 0 \text{ otherwise,} \\ T &= \{t \in Z^{r+} : \text{there is a } p \in \{0, \dots, p^*\}^r \text{ such that } t_i = 2^{p_i}, i = 1, \dots, r\}. \end{aligned}$$

T is the set of all r -vectors whose components are integral powers of 2 up to $\epsilon' \bar{M}_v(I)$. The following algorithm will be shown to be an FPTAS for Π :

1. Determine $\bar{M}_v(I)$ as described. Set $C \leftarrow \emptyset$.
2. For each $t \in T$, let g be a scaling neighbor of f with respect to t and let h be a box constraint neighbor of g with respect to μ . Use \mathcal{A} to solve the instance $I_t = (S, h)$, obtaining the solution set C_t . Set $C \leftarrow C \cup C_t$.
3. Eliminate redundant entries from C , yielding C' , an ϵ -efficient solution to I .

We first show that the algorithm's running time is within that allowed for an FPTAS and then demonstrate its correctness.

Running time: The running time is bounded by the product of the number of instances I_t that are solved and the worst case time to find a C_t . Step 1 can be performed in time $O([L(I)]^{k_5})$, for some $k_5 \in Z^+$. Step 3 can be performed in time $O(|C|^{k_4})$, for some $k_4 \in Z^+$.

Let η be the number of instances I_t that are solved, that is, $\eta = |T| = (1 + p^*)^r$. If $\epsilon' \bar{M}_v(I) < 1$, then $p^* = 0$ and $\eta = 1$. Otherwise,

$$\eta < (1 + \log(\epsilon' \bar{M}_v(I)))^r \leq (2 - \log(n) + \log(\bar{M}_v(I)))^r \in O([L(I)]^{k_1}),$$

for some $k_1 \in Z^+$, since $\epsilon' \leq \frac{2}{n}$, $\log(M_v(I)) \leq L(I)$, and r is fixed.

For any $t \in T$, let τ be the time needed for \mathcal{A} to compute C_t . Then,

$$\tau \in O\left([L(I_t) \times M_v(I_t)]^{k_2}\right) \subseteq O\left([L(I_t) \times \|\mu\|]^{k_2}\right) = O\left([L(I)/\epsilon']^{k_3}\right),$$

for some $k_2, k_3 \in Z^+$. Thus, the total time needed by the algorithm is at most, for some $k \in Z^+$, $\eta \times \tau \in O([L(I)/\epsilon']^k)$. Since $\epsilon' = \frac{\epsilon}{n}$ or $\epsilon' = \frac{2}{n}$, the algorithm runs within the time allowed for an FPTAS.

Accuracy: For the algorithm to be correct, it has to compute a solution $y(x) \in C'$ for each $x \in S$ such that $f(y) \leq (1 + \epsilon)f(x)$ (ϵ -efficient).

For any $t \in T$, the function g satisfies, for $x \in S$:

$$\text{if } t_i = 1 \text{ then } g_i(x) = f_i(x), \quad (3)$$

$$\text{if } t_i > 1 \text{ then } \lfloor \frac{f_i(x)}{t_i} \rfloor - n + 1 \leq g_i(x) \leq \lfloor \frac{f_i(x)}{t_i} \rfloor; \quad (4)$$

and the function h satisfies

$$\text{if } g_i(x) \leq \mu_i \text{ then } h_i(x) = g_i(x), \quad (5)$$

$$\text{if } g_i(x) > \mu_i \text{ then } \mu_i \leq h_i(x) \leq \min\{g_i(x), n\mu_i\}. \quad (6)$$

We will show that for a given $x \in S$, the instance I_t defined as follows will add an appropriate $y(x)$ to the solution set C' . For $i = 1, \dots, r$,

$$t_i = 2^{\lceil \log(\epsilon' f_i(x)) \rceil} \text{ if } \epsilon' f_i(x) \geq 1, \text{ and otherwise } t_i = 1.$$

In particular, if $\epsilon' f_i(x) \geq 1$, then $\frac{\epsilon' f_i(x)}{2} < t_i \leq \epsilon' f_i(x)$.

The reason to use this value for t is that the box constraints μ are redundant when $f(x)$ is scaled by t . To see this, note that for $i = 1, \dots, r$,

$$\left\lfloor \frac{f_i(x)}{t_i} \right\rfloor \leq \frac{f_i(x)}{t_i} < \frac{2}{\epsilon'} < \left\lfloor \frac{4}{\epsilon'} \right\rfloor = \mu_i .$$

Combining the above results yields $g_i(x) < \mu_i$, thus, $h_i(x) = g_i(x)$, and

$$\left\lfloor \frac{f_i(x)}{t_i} \right\rfloor - n + 1 \leq h_i(x) \leq \left\lfloor \frac{f_i(x)}{t_i} \right\rfloor .$$

Now $t \in T$, so I_t is solved in Step 2 and an efficient set of solutions C_t is computed. The set C_t therefore contains a point $y(x)$ such that $h(y) \leq h(x)$. It remains to be shown that $y(x)$ also satisfies $f(y) \leq (1 + \epsilon')f(x)$. Since $\epsilon' \leq \epsilon$, this will complete the proof.

Consider any index $i \in \{1, \dots, r\}$. Then

$$h_i(y) \leq h_i(x) = g_i(x) < \mu_i;$$

so $h_i(y) = g_i(y)$ and

$$\left\lfloor \frac{f_i(y)}{t_i} \right\rfloor - n + 1 \leq h_i(y) \leq \left\lfloor \frac{f_i(y)}{t_i} \right\rfloor .$$

Consider two cases for the value of $\epsilon' f_i(x)$.

Case 1: $\epsilon' f_i(x) \geq 1$. Then

$$\left\lfloor \frac{f_i(y)}{t_i} \right\rfloor \leq h_i(y) + n - 1 \leq h_i(x) + n - 1 \leq \left\lfloor \frac{f_i(x)}{t_i} \right\rfloor + n - 1$$

and so, following the above inequalities,

$$f_i(y) < t_i \left\lfloor \frac{f_i(y)}{t_i} \right\rfloor + t_i \leq t_i \left\lfloor \frac{f_i(x)}{t_i} \right\rfloor + nt_i \leq f_i(x) + nt_i \leq (1 + n\epsilon')f_i(x) \leq (1 + \epsilon)f_i(x) .$$

Case 2: $\epsilon' f_i(x) < 1$. Then

$$f_i(y) = g_i(y) = h_i(y) \leq h_i(x) = g_i(x) = f_i(x) < (1 + \epsilon') f_i(x) \quad . \quad \square$$

In summary, an FPTAS can be constructed from a VPP algorithm for any optimization problem for which the class of objective functions is quasi-closed under both scaling and box constraints. This results in stronger sufficiency conditions than those stated in Papadimitriou and Steiglitz (1982). In particular, this is significant when problems with general integer variables are considered.

Much of the literature on approximation schemes only considers binary domains, that is, 0 – 1 problems. The exceptions generally treat non-binary problems by reduction to the binary case, such as in Ibarra and Kim (1975), Lawler (1979), and Kannan and Korte (1984). A typical reduction considers the binary representation of the general integer variables; this works for separable linear objective functions, but not for the more general objective functions considered later in this paper.

The algorithm in the sufficiency conditions of Papadimitriou and Steiglitz (1982), rather than being VPP, runs in time $O([L(I) \times p_{max}]^{k_1})$, for some $k_1 \in Z^+$. For a binary domain, this is essentially the same as a VPP algorithm. For a general integer domain, though, Theorem 2 is stronger; it guarantees the existence of an FPTAS even if the optimization algorithm uses as much time as $O([L(I) \times p_{max} \times u_{max}]^{k_2})$, for some $k_2 \in Z^+$.

4 Source-To-Tree Network Flow

The Source-to-Tree (STT) Network Flow problem is introduced in this section. This problem is interesting because it generalizes a wide variety of commonly treated discrete (usually single-objective) optimization problems. When proving a property (such as existence of an FPTAS) for a problem, proving it for a more general problem is sometimes easier; we use the STT Network Flow problem in this way. To this end, a new form of reducibility, VPP reducibility, is defined in the Appendix.

We start by introducing the problem and presenting a VPP algorithm for a canonical version of

it. In the remainder of this section and the next section, we use reduction to explore the boundary between easy and difficult versions of the problem.

4.1 The Source-To-Tree Network Flow Feasible Set

This section introduces the notation that we will use to describe STT flow networks. Recall that a *generalized network* is a network in which the flow that enters at the tail of an arc is multiplied by an arc-specific constant before it emerges at the head of the arc. Two directed arcs are said to be *anti-parallel* if they connect the same pair of nodes but are oriented in opposite directions.

Intuitively, an STT network is an $m + 1$ node generalized network that satisfies the following conditions. The subgraph induced by nodes $\{1, \dots, m\}$ is referred to as the *tree part* of the network. The tree part of the network has a special forest-like structure: if each set of parallel and anti-parallel arcs is combined into a single undirected edge, the result is a forest. Node 0 is a source/sink node and is adjacent to some other nodes in such a way that the entire graph is connected. Each node, except the source/sink node, has a demand or supply.

More formally, an STT network is defined by a 5-tuple $G = (N, A, u, \mu, b)$. The set of nodes is $N = \{0, 1, \dots, m\}$ and the set of arcs is $A \subseteq N \times N$. Arcs are directed, and multiple parallel and anti-parallel arcs are permitted. If only a single arc goes from node v to node w , however, it may be represented as (v, w) for convenience. The network is connected and the arcs in A have a particular structure: if the arc directions are removed from the tree part of the network, and each resulting set of parallel edges is replaced by a single edge, the outcome is a forest.

The vector u (of length $|A|$) contains the flow capacity $u_a \in \mathbb{Z}^+$ for each arc $a \in A$, that is, the upper bound on the flow allowed on arc a . The vector μ lists the corresponding flow multipliers $\mu_a \in \mathbb{Z}^{\geq}$; if flow x enters the tail of arc a , then the flow $\mu_a x$ emerges at the head of the arc. The vector b specifies the demand $b_v \in \mathbb{Z}$ at each node $v \in N \setminus \{0\}$. If $b_v > 0$, then v is a *demand* node; if $b_v < 0$, then v is a *supply* node; otherwise $b_v = 0$, and v is a *transshipment* node.

Let $x \in \mathbb{Z}^{|A| \geq}$ be a flow in G . The *excess flow* into node v , for $v \in N \setminus \{0\}$, is the amount by which the flow into v exceeds the sum of the demand at v and the flow out of v , and can be written

as

$$e_v(x) = \sum_{a \in A: a=(w,v)} \mu_a x_a - \sum_{a \in A: a=(v,w)} x_a - b_v .$$

When discussing the excess flow into each node of a set that includes node 0, node 0 is implicitly excluded from consideration unless otherwise indicated. The flow into node v is called

$$\begin{aligned} \text{sufficient} & \quad \text{if } e_v(x) \geq 0, \\ \text{deficient} & \quad \text{if } e_v(x) \leq 0, \text{ or} \\ \text{exact} & \quad \text{if } e_v(x) = 0. \end{aligned}$$

A flow x is said to be *sufficient*, *deficient*, or *exact* if the flow into every node $v \in N \setminus \{0\}$ has the corresponding attribute.

Many classes of STT networks are interesting. A class can be identified using a 5-tuple in which each component specifies one of the following restrictions about the network class.

1. Restrictions on parallel arcs: Parallel arcs may or may not be allowed in the tree part of the network. This restriction refers to the directed arcs in G , not to the undirected edges mentioned in the definition of an STT network. Excluding parallel arcs does not preclude the use of anti-parallel arcs. This restriction does not apply to arcs to or from node 0.
2. Restrictions on arc flow capacities: The arcs of the network can have either unit capacities or general nonnegative integer capacities.
3. Restrictions on arc flow multipliers: Multipliers can be either unity, as in an ordinary network, or general nonnegative integers, as in a generalized network.
4. Restrictions on node demands: A network may consist of all transshipment nodes, all demand nodes, all supply nodes, or a mixture. These restrictions do not apply to node 0.
5. Restrictions on node excesses for feasible flows: A flow may be required to be sufficient, deficient, or exact, or it may be unrestricted. Note that this is a condition on acceptable solutions, whereas the previous conditions refer to characteristics of the graph structure.

In Safer and Orlin (1995b), problem classes are defined using a 5-field “vocabulary” based on these restrictions, similar to $\alpha|\beta|\gamma$ in scheduling theory. In the present paper, though, classes are simply described when they are needed. One class that will be encountered frequently contains typical generalized networks: no parallel arcs, general flow capacities and flow multipliers, a mixture of different kinds of node demands, and sufficient flows. In the remainder of this paper we use \mathcal{C} to denote this network.

Definition: A flow $x \in Z^{|A| \geq}$ is *feasible* if it satisfies the restrictions on node excesses for feasible flows and if, in addition, $x \leq u$.

The set of feasible flows for an instance constitutes the feasible set S for that instance; the integer program that specifies the set is generally not written out explicitly. The collection of feasible sets S for the networks in a class of STT networks is the family of feasible sets Ξ for an STT Network Flow problem defined on the class.

In order to specify an STT Network Flow optimization or feasibility problem completely, a family \mathcal{F} of r -criteria objective functions defined on the vectors x of flows must be specified; these are discussed in the next subsection. For a feasibility problem, an r -dimensional target vector must be included as well.

4.2 Objective Functions

The problems discussed in the remainder of this paper are distinguished from each other by the structures of their feasible regions and by their objective functions. This section examines several objective functions that arise frequently in practice.

Many common objective functions are monotonically increasing or decreasing. A function $f : Z^{n \geq} \rightarrow Z^{r \geq}$ is *monotonically increasing* or *order preserving* if for $x, y \in Z^{n \geq}$, $x \leq y$ implies that $f(x) \leq f(y)$. A function f is *monotonically decreasing* or *order reversing* if the function $g = -f$ is monotonically increasing, and f is *monotonic* or *order monotone* if it is either monotonically increasing or monotonically decreasing. \mathcal{AP}_r , \mathcal{AR}_r , and \mathcal{AM}_r represent the sets of additively-separable, r -criteria functions that are, respectively, order preserving, order reversing, and order

monotone.

The necessary and sufficient conditions described in sections 3.1 and 3.2 for the existence of an FPTAS apply to problems with quite general objective functions. Often, though, VPP algorithms use dynamic programming, which typically requires separable objective functions; this is so that the value of the objective function can be determined incrementally by optimizing the value of one variable at a time. The most common separable objective functions are additively separable, that is, they can be written as $f(x) = \sum_{j=1}^n f_j(x_j)$. Functions of the form $f(x) = \prod_{j=1}^n f_j(x_j)$ and $f(x) = \min\{f_j(x_j), j = 1, \dots, n\}$ are also used. All objective functions used in the remainder of this paper are additively separable, even when this is not mentioned explicitly.

4.3 Some Simple STT Network Flow Problems and a VPP Algorithm

A network in \mathcal{C} (the set of typical generalized networks with sufficient flows defined in Section 4.1) is said to be in canonical form if a single arc connects node 0 to each other node, each other connected pair of nodes is connected by a single pair of anti-parallel arcs, and the tree part of the network is connected. More formally, a network $G \in \mathcal{C}$ is in *canonical form* if it satisfies the following conditions:

1. $\forall v \in N \setminus \{0\}$, A contains a single arc $(0, v)$ and no arc $(v, 0)$.
2. The tree part of G is connected. If nodes v and w are adjacent, then A contains a single arc (v, w) and a single arc (w, v) .

The subclass of networks of \mathcal{C} in canonical form is denoted by \mathcal{C}' .

A tree T , distinct from but related to the tree part of G , is associated with G for the purpose of describing the VPP algorithm. The tree T has nodes $\{1, \dots, m\}$; nodes v and w of T are connected by an arc if and only if nodes v and w are adjacent in G . The directions of the tree arcs are specified next.

In a postorder traversal of a tree, each subtree of the root is traversed (recursively, in postorder) and then the root is visited (Knuth, 1973, Johnson and Niemi, 1983). Choose an arbitrary node ρ

as the root of T . Label the nodes according to any postorder ordering with respect to the root ρ , representing the label of node v by $l(v)$. In particular, $l(\rho) = n$. The VPP dynamic programming algorithm to be described processes the nodes of $N \setminus \{0\}$ in increasing label order.

Direct all the arcs of T away from ρ and denote by d_v the number of children of node v in T . This number is referred to as the *outdegree* of v . For each node v , let $T[v, j]$, $j = 1, \dots, d_v$, be the subtree of T defined by v , the first j children of v , and all the descendants of those children, where the children are ordered by label value. That is, $T[v, 0]$ is just the node v and $T[\rho, d_\rho] = T$. Let $G[v, j]$ be the subgraph of G defined by node 0 and the nodes of G that correspond to the nodes of $T[v, j]$.

Next, a VPP algorithm that solves the feasibility problem on STT networks in canonical form with sufficient flow (i.e., networks in \mathcal{C}') is presented. For the purpose of stating the algorithm, we say that a flow x in G is *feasible* with respect to $G[v, j]$ if it satisfies the following three conditions:

1. The flow into each node of $G[v, j]$, except possibly into node v itself, is sufficient.
2. If $(w, w') \notin G[v, j]$, then $x_{(w, w')} = 0$.
3. $x \in Z^{|A| \geq}$ and $x \leq u$.

The key component in developing the VPP algorithm is the computation of the maximum flow into node v over all flows x that are both feasible with respect to $G[v, j]$ and have $f(x) = k$, for a given $k \in Z^{r+}$. This flow is denoted as $\Theta(v, j, k)$; our interest in this quantity is motivated by the following lemma.

Lemma 1: Consider the problem $\Pi = (\mathcal{C}', \mathcal{AP}_1, Feas)$, but in which we want to find a solution x that hits the target value exactly (i.e., with $f(x) = M$ rather than $f(x) \leq M$). Let $I = (G, f, M)$ be an instance of Π . Then I has a solution if and only if $\Theta(\rho, d_\rho, M) \geq 0$.

Proof: We first prove that the existence of a solution x for the instance I implies that $\Theta(\rho, d_\rho, M) \geq 0$. So let x be a feasible flow. Then,

1. The flow into each node of $G[\rho, d_\rho]$ is sufficient (including the flow into ρ).

2. All arcs are in $G[\rho, d_\rho]$.

3. $x \in Z^{|A| \geq}$ and $x \leq u$.

This implies that x is feasible with respect to $G[\rho, d_\rho]$ and that $f(x) = M$. Since the flow into ρ is sufficient, $e_\rho(x) \geq 0$. Given that $\Theta(\rho, d_\rho, M)$ is the maximum of a set that contains $e_\rho(x) \geq 0$, it must be the case that $\Theta(\rho, d_\rho, M) \geq 0$.

In the other direction, suppose that $\Theta(\rho, d_\rho, M) \geq 0$. This implies the existence of a flow x that is feasible with respect to $G[\rho, d_\rho] = G$ and satisfies $f(x) = M$. Since $e_\rho = \Theta(\rho, d_\rho, M) \geq 0$, it follows that the flow into every node is sufficient. Therefore x is a feasible flow that solves the instance I . \square

The computation of $\Theta(v, j, k)$ is described below. The details depend on whether $j = 0$ or $j > 0$. When $j = 0$, $\Theta(v, 0, k)$ can be computed by solving the following program.

$$\Theta(v, 0, k) = \max_{x_{0v}} \mu_{0v} x_{0v} - b_v \quad (7)$$

$$\text{s.t.} \quad f_{0v}(x_{0v}) = k \quad (8)$$

$$x_{0v} \leq u_{0v} \quad (9)$$

$$x_{0v} \in Z^{\geq} \quad (10)$$

When $j > 0$, let w be the j^{th} child of v and solve the following program.

$$\Theta(v, j, k) = \max_{\alpha, \beta, \gamma, \delta, x_{vw}, x_{wv}} \Theta(v, j-1, \alpha) + \mu_{wv} x_{wv} - x_{vw} \quad (11)$$

$$\text{s.t.} \quad \Theta(w, d_w, \beta) + \mu_{vw} x_{vw} - x_{wv} \geq b_w \quad (12)$$

$$f_{vw}(x_{vw}) = \gamma \quad (13)$$

$$f_{wv}(x_{wv}) = \delta \quad (14)$$

$$\alpha + \beta + \gamma + \delta = k \quad (15)$$

$$x_{vw} \leq u_{vw} \quad (16)$$

$$x_{wv} \leq u_{wv} \quad (17)$$

$$\alpha, \beta, \gamma, \delta, x_{vw}, x_{wv} \in Z^{\geq} \quad (18)$$

A VPP algorithm to solve this problem is presented as Algorithm 1.

Algorithm 1: The algorithm to solve the canonical STT problem in which the target value must be reached exactly

for $i = 1, \dots, n$

 Let v be the node with $l(v) = i$

for $j = 0, \dots, d_v$

for $k = 0, \dots, M$

 Compute $\Theta(v, j, k)$

if $\Theta(\rho, d_\rho, M) \geq 0$ **then** x is the desired solution, **else** the problem is infeasible

Theorem 3: Algorithm 1 is VPP for the canonical feasibility problem $\Pi = (\mathcal{C}', \mathcal{AP}_1, Feas)$ in which we want to find a solution x that hits the target value exactly.

Proof: The proof has two parts: the algorithm's correctness and its running time.

Correctness: In the case that $j = 0$, the maximum excess on $G[v, 0]$ must be computed. This is just the subgraph of G induced by the nodes 0 and v , which, because $G \in \mathcal{C}'$, contains a single arc $(0, v)$. The quantity $\mu_{0v}x_{0v} - b_v$ is the excess flow into node v on $G[v, 0]$ when the flow on arc $(0, v)$ is x_{0v} . The program specified by equations (7)-(10) maximizes this excess subject to the feasibility conditions, and so correctly computes $\Theta(v, 0, k)$.

In the case that $j > 0$, the maximum excess on $G[v, j]$ must be computed. Note that the nodes are considered in increasing label order, the postorder label of a node is greater than the labels of its children, and the children of a node are processed in increasing label order; therefore the values of $\Theta(\cdot, \cdot, \cdot)$ that appear in equations (11) and (12) are computed before they are used.

The structure of equations (11)-(18) comes from decomposing the flow in $G[v, j]$ and the value k into four parts: the flow in $G[v, j - 1]$, with value α ; the flow in $G[w, d_w]$, with value β ; and the flows on arcs (v, w) and (w, v) , with values γ and δ . The total required value of k can be partitioned this way because additively separable objective functions are used.

The objective function, equation (11), expresses the value of the excess flow into node v , given the flows on arcs (v, w) and (w, v) and in the first $(j - 1)$ subtrees of node v , and subject to the flow in $G[v, j - 1]$ having value α . This is maximized to determine $\Theta(v, j, k)$.

Equation (12) ensures that the flow into node w is sufficient; recall that it need not have been sufficient in $G[w, d_w]$, and anyhow the arcs (v, w) and (w, v) are not in $G[w, d_w]$. The value contributed by the flow in $G[w, d_w]$ is constrained to be β . Since equation (12) expresses the feasibility of the flow, the excess might as well be as large as possible, so $\Theta(w, d_w, \beta)$ is assumed.

Equations (13) and (14) constrain the flow values on arcs (v, w) and (w, v) . Equation (15) ensures that the four parts of the flow add up to k , and equations (16)-(18) are needed to guarantee feasibility.

So the program specified by equations (11)-(18) determines $\Theta(v, j, k)$. The final value computed by the algorithm is $\Theta(\rho, d_\rho, M)$. If the value is non-negative, then Lemma 1 shows that the value of x found by the algorithm is a solution to the instance. Similarly, if $\Theta(\rho, d_\rho, M) < 0$, then Lemma 1 implies that the instance has no solution. Therefore the algorithm is correct.

Running time: The value of $\Theta(v, j, k)$ is computed only $M \times |A|$ times. The rest of this proof shows that each computation can be performed in VPP time. This yields the conclusion that the algorithm runs in VPP time.

A computation with $j = 0$ requires solving the program expressed by (7)-(10). For a node v , this can be solved with $O(\log(u_{0v} + 1))$ evaluations of $f_{0v}(\cdot)$ using binary search over possible values of x_{0v} , because f_{0v} is monotonic and additively separable.

A computation with $j > 0$ requires solving the program expressed by (11)-(18). Only $O(k^3)$ distinct 4-tuples $(\alpha, \beta, \gamma, \delta)$ satisfy equations (15)-(18), so if the program can be solved for each 4-tuple in VPP time, then the entire algorithm will run within the allowed time.

Fix values of α, β, γ , and δ . The value of x_{vw} must satisfy

$$f_{vw}(x_{vw}) = \gamma \tag{19}$$

$$x_{vw} \leq u_{vw} \tag{20}$$

$$x_{vw} \in Z^\geq \tag{21}$$

Let λ_{vw} and ξ_{vw} be the smallest and the largest values that satisfy equations (19)-(21); they can be

found with $O(\log(u_{vw} + 1))$ evaluations of the $f_{vw}(\cdot)$ by binary search. Similarly, the corresponding values of λ_{wv} and ξ_{wv} for x_{wv} can be found with $O(\log(u_{wv} + 1))$ evaluations of the $f_{wv}(\cdot)$. The following program yields the optimal values of x_{vw} and x_{wv} , given $(\alpha, \beta, \gamma, \delta)$.

$$\max_{x_{vw}, x_{wv}} \quad \mu_{vw} x_{vw} - x_{vw} \quad (22)$$

$$\text{s.t.} \quad \mu_{vw} x_{vw} - x_{vw} \geq b_w - \Theta(w, d_w, \beta) \quad (23)$$

$$\lambda_{vw} \leq x_{vw} \leq \xi_{vw} \quad (24)$$

$$\lambda_{wv} \leq x_{wv} \leq \xi_{wv} \quad (25)$$

$$x_{vw}, x_{wv} \in Z^{\geq} \quad (26)$$

To see that this program can be solved in VPP time, consider the following cases:

1. $\mu_{vw} = 0$: Set $x_{vw} = \lambda_{vw}$. If $\lambda_{vw} > \mu_{vw} x_{vw} + \Theta(w, d_w, \beta) - b_w$, then the program is infeasible.

Otherwise, set

$$x_{vw} = \min\{\xi_{vw}, \mu_{vw} x_{vw} + \Theta(w, d_w, \beta) - b_w\}.$$

2. $\mu_{vw} = 0, \mu_{vw} \geq 1$: Set $x_{vw} = \lambda_{vw}$. If $[b_w - \Theta(w, d_w, \beta) + x_{vw}]/\mu_{vw} > \xi_{vw}$, then the program is infeasible. Otherwise, set,

$$x_{vw} = \max\{\lambda_{vw}, [b_w - \Theta(w, d_w, \beta) + x_{vw}]/\mu_{vw}\}.$$

3. $\mu_{vw}, \mu_{wv} \geq 1$: Suppose that, in some feasible solution of equations (22)-(26), both $x_{vw} < \xi_{vw}$ and $x_{wv} < \xi_{wv}$. Incrementing each variable by one would yield a new solution with objective value at least as good as the value of the previous solution. Therefore, attention can be restricted to solutions in which at least one of these variables is at its upper bound. Each case is considered and the one with the largest objective function value is used.

(a) $x_{vw} = \xi_{vw}$: Set x_{wv} as in case 1.

(b) $x_{wv} = \xi_{wv}$: Set x_{vw} as in case 2.

□

Apply Lemmas A3 and A4 (in the Appendix) to Theorem 3 in order to conclude that the optimization problem corresponding to the canonical feasibility problem $\Pi = (\mathcal{C}', \mathcal{AP}_1, Feas)$ in which we want to find a solution x that hits the target value exactly has an FPTAS.

Corollary 1: The optimization problem $(\mathcal{C}', \mathcal{AP}_1, Opt)$ in which we want to find all possible function values has an FPTAS.

4.4 Using Reduction

The previous section presented an explicit VPP algorithm. Appendix A describes VPP reductions between problems which allow us to establish the existence of an FPTAS without explicitly specifying the algorithm. The reductions can also be used to show that problems do not have a VPP algorithm—that they are difficult. We use reductions to generalize the previous results from \mathcal{C}' to \mathcal{C} .

Lemma 2: Let $\Pi = (\mathcal{C}, \mathcal{AP}_1, Feas)$ be an STT problem in which we want to hit the target value exactly and $\Pi' = (\mathcal{C}', \mathcal{AP}_1, Feas)$ be the same problem on canonical-form networks. Then $\Pi \equiv_{VPP} \Pi'$.

Proof:

$\Pi' \propto_{VPP} \Pi$: $\mathcal{C}' \subset \mathcal{C}$, so the identity mapping is a reduction from Π' to Π .

$\Pi \propto_{VPP} \Pi'$: Let $I = (S, f, M) \in \Pi$ be given, where S is defined by a graph $G = (N, A, u, \mu, b) \in \mathcal{C}$. Define $I' = (S', f', M) \in \Pi'$, where S' is defined by the graph $G' = (N', A', u', \mu', b') \in \mathcal{C}'$. The graph G' is the same as G , except as follows:

1. (Eliminate arcs into node 0) For each arc $a = (v, 0) \in A$:
 - (a) Delete arc a from A' .
 - (b) Add a node w_a to N' with $b_{w_a} = 0$.
 - (c) Add an arc $a' = (v, w_a)$ to A' with $u_{a'} = u_a$, $\mu_{a'} = \mu_a$, and $f'_{1,a'}(x_{a'}) \equiv f_{1,a}(x_a)$.
2. (Eliminate parallel arcs from node 0) For each node $v \in \{1, \dots, m\}$, if the graph contains multiple arcs from node 0 to node v , do the following steps for all but one of the arcs. Let $a = (0, v)$ be the

arc on which the steps are performed.

- (a) Delete arc a from A' .
 - (b) Add a node w_a to N' with $b_{w_a} = 0$.
 - (c) Add an arc $a' = (0, w_a)$ to A' with $u_{a'} = u_a, \mu_{a'} = 1$, and $f_{1,a'}(x_{a'}) \equiv 0$.
 - (d) Add an arc $a'' = (w_a, v)$ to A' with $u_{a''} = u_a, \mu_{a''} = \mu_a$, and $f'_{1,a''}(x_{a''}) \equiv f_{1,a}(x_{a''})$.
3. (Guarantee an arc from node 0 to each other node) For each node $v \in N$ such that arc $(0, v) \notin A$: Add an arc $a' = (0, v)$ to A' with $u_{a'} = 0, \mu_{a'} = 1$, and $f'_{1,a'}(x_{a'}) \equiv 0$.
4. (Guarantee pairs of anti-parallel arcs) For each arc $(v, w) \in A$ such that arc $(w, v) \notin A$: Add an arc $a' = (w, v)$ to A' with $u_{a'} = 0, \mu_{a'} = 1$, and $f'_{1,a'}(x_{a'}) \equiv 0$.
5. (Guarantee connectedness) If the subgraph of G induced by nodes $\{1, \dots, n\}$ is not connected:
- (a) Add a node w to N' with $b_w = 0$.
 - (b) Add pairs of arcs $a' = (v, w)$ and $a'' = (w, v)$ to A' , with $u_{a'} = u_{a''} = 0, \mu_{a'} = \mu_{a''} = 1$, and $f'_{1,a'}(x_{a'}) \equiv f'_{1,a''}(x_{a''}) \equiv 0$, to at least one node v in each component of the subgraph.

For each arc a that is in both A and A' , define $f'_{1,a}(x_a) \equiv f_{1,a}(x_a)$; then $f' \in \mathcal{AP}_1$. By construction, $G' \in \mathcal{C}'$; furthermore, G' can be constructed in VPP time, and $L(I)$ is polynomial in $L(I')$. The reduction consists of constructing G' and solving I' as described in the previous section. This yields a solution of I directly using the correspondences described in the construction of G' . \square

Theorem 3 and Lemma 2 together with Lemma A3 in the Appendix yield the following corollary.

Corollary 2: The feasibility problem $(\mathcal{C}, \mathcal{AP}_1, Feas)$ in which we want to hit the target exactly has a VPP algorithm and the corresponding optimization problem $(\mathcal{C}, \mathcal{AP}_1, Opt)$ in which we want to find all possible function values has an FPTAS.

5 Easy Problems and Hard Problems

Many problems can be defined on STT networks. This section explores the boundary between “easy” and “hard” problems. The easy problems are those for which an FPTAS exists; the hard problems are those for which no VPP algorithm can exist unless $P = NP$.

The proofs of hardness use the Subset Sum problem, which is NP -complete (Garey and Johnson, 1979).

Subset Sum

Instance: (n, V, v) . The number of items n ; the target sum V ; and the sizes $v_j, j = 1, \dots, n$, are positive integers such that $v_j \leq V, j = 1, \dots, n$.

Question: Is there a set $S \subseteq \{1, \dots, n\}$ such that $\sum_{j \in S} v_j = V$?

Theorem 4: Consider a set of STT networks \mathcal{C}_1 that contains parallel arcs, general flow capacities, unit flow multipliers, transshipment nodes, and sufficient flows. Let \mathcal{C}_2 be a similar set of networks except with deficient flows. Let $\Pi_1 = (\mathcal{C}_1, \mathcal{AP}_1, Feas)$ and $\Pi_2 = (\mathcal{C}_2, \mathcal{AP}_1, Feas)$ be problems in which we want to hit the target value exactly. Then neither Π_1 nor Π_2 has a VPP algorithm unless $P = NP$.

Proof: The theorem will be proved by reducing Subset Sum to each of the STT problems in such a way that a VPP algorithm for one of the latter would yield a polynomial-time algorithm for the former. The problems need not be multi-objective—the single objective versions are sufficient for the proof. Since Subset Sum is NP -complete, such VPP algorithms cannot exist unless $P = NP$.

First consider the problem Π_1 . Given an instance (n, V, v) of Subset Sum, construct an STT network $G = (N, A, U, \mu = 1, b)$ as follows. The set N has three nodes, and both non-source nodes are transshipment nodes; that is, $N = \{0, 1, 2\}$, with $b_1 = b_2 = 0$. The arcs are described in the

following table:

j	Arc	Capacity	Multiplier	Value $f_j(x_j)$
$1, \dots, n$	$(1, 2)$	v_j	1	$2n + 2$ if $x_j = v_j$ 2 if $0 < x_j < v_j$ 0 if $x_j = 0$
$n + 1$	$(0, 1)$	V	1	0
$n + 2$	$(2, 0)$	V	1	1 if $x_j = V$ 0 otherwise

The objective function is $f(x) = \sum_{i=1}^{n+2} f_j(x_j)$. Since the flows into nodes 1 and 2 are sufficient, the total flow on arcs 1 through n must be exactly V . So the instance of Subset Sum has a solution if and only if at least one instance of $(G, f, k(2n + 2) + 1)$, $k = 0, \dots, n$, of Π_1 has a solution, and the solution to any of these instances immediately yields a solution to the instance of Subset Sum. Furthermore, a VPP algorithm for the STT problem is polynomial-time in the size of the Subset Sum problem.

The proof for the deficient flow case is similar, except that the objective functions on arcs $n + 1$ and $n + 2$ are interchanged. \square

5.1 Exact Flows

This section considers problems in which the flows must be exact, that is, the excess flow into each node must be zero. Theorem 5 identifies some such problems that can be solved by an FPTAS. Theorems 6 and 7 show that this result cannot be extended to graphs with both supply and demand nodes or to problems with objective functions that are not order-preserving.

Theorem 5: Let \mathcal{C}_1 be a set of STT networks with no parallel arcs in the tree part of the network, general flow capacities, unit flow multipliers, only demand nodes (i.e., $b \geq 0$), and exact flow requirements. Let \mathcal{C}_2 be a similar set of networks but with only supply nodes (i.e., $b \leq 0$). Let $\Pi_1 = (\mathcal{C}_1, \mathcal{AP}_r, Opt$ and $\Pi_2 = (\mathcal{C}_2, \mathcal{AP}_r, Opt$ for some $r \in Z^+$. Then Π_1 and Π_2 each have an FPTAS.

Proof: Consider the problem Π_1 . Define the STT network feasible set \mathcal{C}_3 to be the same as \mathcal{C}_1 but with sufficient flows and the problem $\Pi_3 = (\mathcal{C}_3, \mathcal{AP}_r, Feas)$. Note that $\mathcal{C}_3 \subset \mathcal{C}$, so Π_3 has an FPTAS. Next, we prove by reduction that $\Pi_1 \propto_{VPP} \Pi_3$.

Suppose that an instance $I_1 = (S_1, f) \in \Pi_1$ is specified, where $S_1 \in \mathcal{C}_1$ is defined by a graph G . Let $S_3 \in \mathcal{C}_3$ also be defined by G and define the instance $I_3 = (S_3, f, M) \in \Pi_3$ for any $M \in \mathbb{Z}^{r+}$.

Consider any flow x that solves I_3 . This flow can be decomposed into flows around circuits and flows on chains (simple directed paths) from supply nodes to demand nodes (see Ford and Fulkerson, 1962, or Ahuja, Magnanti, and Orlin, 1993). Suppose that the flow around some circuit in such a decomposition is non-zero; since f is order-preserving, no component of the objective would be made worse by eliminating the flow around the circuit. So the circuits in the decomposition can be assumed to have zero flow.

All nodes except node 0 are demand nodes, so all chains in the decomposition are from node 0 to another node. For each node v with positive excess, that is, $e_v(x) > 0$, consider the chains from node 0 to node v . Reducing the flow on one or more of these chains yields a flow x' with $e_v(x') = 0$. As with the circuits, no component of the objective function will be made worse by reducing the flow in this way. Reduce the flow in this way for each node to obtain an exact flow \bar{x} . The flow \bar{x} solves I_1 so $\Pi_1 \propto_{VPP} \Pi_3$.

The proof for Π_2 is substantially the same. \square

A stronger version of Theorem 5, which addresses simultaneous minimization in some criteria and maximization in others, is proved in Safer and Orlin (1995b). The next theorem shows that requiring an exact flow in a network that contains both supply and demand nodes results in a hard problem, even in the case of a single objective function.

Theorem 6: Let \mathcal{C}_1 be a set of STT networks with no parallel arcs, general flow capacities, unit flow multipliers, both supply and demand nodes, and exact flow requirements. Then no VPP algorithm exists for the feasibility problem $\Pi = (\mathcal{C}_1, \mathcal{AP}_1, Feas)$ unless $P = NP$.

Proof: The proof is by reduction of Subset Sum to Π .

Given an instance (n, V, v) of Subset Sum, construct an STT network $G = (N, A, u, \mu, b)$ as follows. The set N has $n + 2$ nodes $\{0, 1, \dots, n, n + 1\}$. It has a node j for each item, with supply $b_j = -v_j$, $j = 1, \dots, n$. Node $(n + 1)$ has demand V , that is, $b_{n+1} = V$. The set A has $2n$ arcs, two originating at each node j : $\alpha_j = (j, 0)$ and $\beta_j = (j, n + 1)$, $j = 1, \dots, n$. Arcs α_j and β_j have capacity v_j , and the objective function $g(x_k) = 1$ for $x_k > 0$, and zero otherwise. The objective function $f(x) = \sum_{j=1}^n [g(x_{\alpha_j}) + g(x_{\beta_j})]$ is order-preserving.

It is clear from the construction that an instance of Subset Sum has a solution if and only if the instance (G, f, n) of Π has a solution, and that a VPP algorithm for the STT problem is polynomial-time in the size of the Subset Sum instance. \square

The next theorem shows that if we try to maximize criteria that are monotonically increasing on networks that contain only transshipment nodes, then the corresponding exact flow feasibility problem is hard even for a single objective.

Theorem 7: Let \mathcal{C}_1 be an STT network with no parallel arcs in the tree part of the network, general flow capacities, unit flow multipliers, only transshipment nodes (i.e., $b = 0$), and exact flow requirements. Then no VPP algorithm exists for the problem $\Pi = (\mathcal{C}_1, \mathcal{AP}_1, Feas)$ in which we want to maximize the value of the objective function unless $P = NP$.

Proof: The proof follows from a direct reduction from Subset Sum.

Given an instance (n, V, v) of Subset Sum, an STT network $G = (N, A, u, \mu, b)$ is constructed as follows. The set N has $n + 2$ nodes $\{0, 1, \dots, n, n + 1\}$. All nodes are transshipment nodes, that is, $b_j = 0$, $j = 1, \dots, n + 1$. The set A has $3n + 1$ arcs that are described in the following table:

Arc	Capacity	Multiplier	Value $f_k(x_k)$
$(0, j), \quad j = 1, \dots, n$	v_j	1	0
$(j, 0), \quad j = 1, \dots, n$	v_j	1	1 if $x_k = v_j$
$(j, n + 1), \quad j = 1, \dots, n$			0 otherwise
$(n + 1, 0)$	V	1	1 if $x_k = V$ 0 otherwise

The objective function $f(x) = \sum_{k \in A} f_k(x_k)$ is order-preserving (with respect to minimization). However, if we are maximizing the objective function, then from the above construction, the instance $(G, f, n + 1)$ of Π has a solution if and only if the Subset Sum instance has a solution. \square

6 STT Networks for Some Well-Known Problems

This section summarizes the implications of the above results about STT Network Flow problems for some well-known discrete optimization problems. In each case, the correspondence is shown by indicating the appropriate STT network structure. The problems treated in this section include the Arborescent Knapsack problem, the Partially-Ordered Knapsack problem, and the Maximum and Minimum Job Sequencing with Due Dates problems. The next section examines some production planning problems in more detail.

6.1 The Aborescent Knapsack Problem

In the Aborescent Knapsack problem, items are grouped into sub-knapsacks, or stuffsacks. Stuffsacks may be nested within other stuffsacks and each stuffsack has its own capacity bound. The structure is called aborescent because the “nested within” relationship on the stuffsacks can be represented by a directed tree, or aborescence. We start by defining the arborescent property.

Definition: A collection of sets $\mathcal{B} = \{B_1, \dots, B_m\}$ is called *arborescent* if for each pair $i, j \in \{1, \dots, m\}$, with $i \neq j$, one of the following relations holds: $B_i \subset B_j$, $B_j \subset B_i$, or $B_i \cap B_j = \emptyset$. \mathcal{B} is sometimes called a *collection of stuffsacks*.

The Aborescent Knapsack feasible set may be defined formally as follows:

Integer Aborescent Knapsack Feasible Set

Instance: $(n, m, \mathcal{B}, V, v, u)$. The number of items n and number of stuffsacks m are non-negative integers. The collection of stuffsacks \mathcal{B} is an aborescent collection of m sets $B_i \subseteq \{1, \dots, n\}, i = 1, \dots, m$, with $B_m = \{1, \dots, n\}$. The stuffsack volumes $V_i, i = 1, \dots, m$; the item volumes $v_j, j = 1, \dots, n$; and the item quantity upper bounds $u_j, j = 1, \dots, n$; are non-negative integers such

that if $j \in B_i$, then $v_j \leq V_i$, $i = 1, \dots, m$, $j = 1, \dots, n$.

Feasible Set: $S(n, m, \mathcal{B}, V, v, u)$ is the set of all $x \in Z^{n+}$ that satisfy the following constraints:

$$\begin{aligned} \sum_{j \in B_i} v_j x_j &\leq V_i, \quad i = 1, \dots, m \\ x_j &\leq u_j, \quad j = 1, \dots, n. \end{aligned}$$

The nesting property of the sets B_j gives the multiple constraints a special structure that allows the Aborescent Knapsack problem to have an FPTAS, despite the difficulty of the Multi-Dimensional Knapsack problem, of which it is a special case. An FPTAS that uses $O(n^3/\epsilon^2)$ time for the binary version of this problem with a single linear, additively-separable objective is presented by Gens and Levner (1979). Results in this paper generalize the result of Gens and Levner to the integer case, in which an item may appear more than once in the knapsack (up to u_j copies of item j may be used). In addition, the present results accommodate multiple criteria.

We present a VPP reduction from the Integer Aborescent Knapsack Feasible Set problem to a special case of the STT Network Flow problem and state a resulting corollary.

Let $\Pi_1 = (\Psi, \mathcal{AM}(r), Feas)$ be the Integer Arborescent Knapsack problem, where Ψ is the collection of Integer Aborescent Knapsack feasible sets and $r \in Z^+$. Suppose an instance $I_1 = (S, f, M)$ of Π_1 is given, with $S = S(n, m, \mathcal{B}, V, v, u)$. Assume that the sets B_i are “topologically” ordered, so that $B_i \subset B_k$ implies that $i < k$; such an ordering can be found quickly (Ahuja, Magnanti, and Orlin, 1993). Define the “first containing set” function ϕ for items by

$$\phi(j) = \min\{k : j \in B_k\}, \quad j = 1, \dots, n$$

and a similar function σ for sets by $\sigma(i) = \min\{k : B_i \subset B_k\}$, $i = 1, \dots, m - 1$, and $\sigma(m) = 0$.

Let \mathcal{C}_2 be the set of STT networks with no parallel arcs in the tree part of the network, general flow constraints, general flow multipliers, all transshipment nodes, and deficient flow requirements, and let $\Pi_2 = (\mathcal{C}_2, \mathcal{AM}_r, Feas)$. Define an instance $I_2 = (C_2, f', M)$ of Π_2 , where C_2 is defined by an STT network $G = (N, A, u, \mu, b)$. The set N has $(n + m + 1)$ nodes: a node α_j corresponding to each item j , a node β_i corresponding to each stuffsack i , and node 0. All nodes are transshipment

nodes. The set A has $(2n + m)$ arcs described in the following table:

Arc	Capacity	Multiplier	Value $f'_k(x_k)$
$(0, \alpha_j), \quad j = 1, \dots, n$	u_j	v_j	$f_j(x_j)$
$(\alpha_j, \beta_{\phi(j)}), \quad j = 1, \dots, n$	$v_j u_j$	1	0
$(\beta_i, \beta_{\sigma(i)}), \quad i = 1, \dots, m$	V_i	1	0

If I_1 has a solution: Suppose that x is a feasible solution to I_1 . One feasible solution to I_2 is shown in the following table:

Arc	Flow
$(0, \alpha_j), \quad j = 1, \dots, n$	x_j
$(\alpha_j, \beta_{\phi(j)}), \quad j = 1, \dots, n$	$v_j x_j$
$(\beta_i, \beta_{\sigma(i)}), \quad i = 1, \dots, m$	$\sum_{j \in B_i} v_j x_j$

This is a feasible flow that has the same objective function value as does the solution of I_1 .

If I_2 has a solution: Suppose that y is a feasible flow for I_2 and consider the following assignments for the knapsack variables:

$$x_j = y_{0\alpha_j}, \quad j = 1, \dots, n \quad . \quad (27)$$

The capacity constraints on arc $(0, \alpha_j)$ guarantee that $x_j \leq u_j, j = 1, \dots, n$. It will be shown, by induction, that $\sum_{j \in B_i} v_j x_j \leq y_{\beta_i \beta_{\sigma(i)}}, i = 1, \dots, m$. Then from the capacity constraints on the arcs $(\beta_i, \beta_{\sigma(i)})$ it follows that $\sum_{j \in B_i} v_j x_j \leq V_i$, so x is a feasible solution for I_1 with the same objective function as y has in I_2 .

$$i = 1 : \quad \sum_{j \in B_1} v_j x_j = \sum_{j \in B_1} v_j y_{0\alpha_j} \leq \sum_{j \in B_1} y_{\alpha_j \beta_{\phi(j)}} = \sum_{j \in B_1} y_{\alpha_j \beta_1} \leq y_{\beta_1 \beta_{\phi(1)}} \quad .$$

The first equality follows from (27), the inequality from the nodes' deficient excess, and the equality

from topological ordering, leading to the last inequality based on deficient excess at node β_1 .

$$\begin{aligned}
i > 1 : \quad \sum_{j \in B_i} v_j x_j &= \sum_{j: \phi(j)=i} v_j x_j + \sum_{k: \sigma(k)=i} \sum_{j \in B_k} v_j x_j \\
&\leq \sum_{j: \phi(j)=i} y_{\alpha_j \beta_i} + \sum_{k: \sigma(k)=i} \sum_{j \in B_k} v_j x_j \\
&\leq \sum_{j: \phi(j)=i} y_{\alpha_j \beta_i} + \sum_{k: \sigma(k)=i} \sum_{j \in B_k} y_{\beta_k \beta_i} \\
&\leq y_{\beta_i \beta_{\sigma(i)}}
\end{aligned}$$

The aborescent network structure gives the first equality. The first inequality follows because the nodes have deficient excess. The next inequality is obtained by induction and the last one is true because node β_i has deficient excess.

Corollary 3: The Integer Aborescent Knapsack multi-criteria optimization problem with only order-monotone criteria has an FPTAS.

6.2 The Maximum Job Sequencing with Due Dates Problem

A special case of the Aborescent Knapsack problem is the Triangular Knapsack problem, in which the stuffsacks are nested sequentially. More precisely, the Triangular Knapsack problem is an Aborescent Knapsack problem in which $m = n$ and $B_i = \{1, \dots, i\}$. One reason for interest in this problem is that a common scheduling problem can be formulated as a Triangular Knapsack problem. Details about related scheduling problems can be found in Graham et al. (1979).

In the Max Job Sequencing with Due Dates problem, a set of independent jobs must be scheduled on a single machine. Those completed by their due dates earn a profit, but those completed late earn nothing; the objective is to earn as much as possible. This can be described more formally as follows:

Max Job Sequencing with Due Dates (Max JSD), Version 1

Instance: (n, p, t, d, K) . The number of jobs n ; the profits p_j , $j = 1, \dots, n$; the execution times t_j , $j = 1, \dots, n$; the due dates d_j , $j = 1, \dots, n$; and the minimum profit target K ; are non-negative

integers such that $t_j \leq d_j$, $j = 1, \dots, n$, and $d_j \leq d_{j+1}$, $j = 1, \dots, n - 1$.

Question: Is there a schedule of jobs with total profit at least K , in which job j earns profit p_j if the job is completed before time d_j and earns 0 profit otherwise?

An FPTAS that runs in time $O(n^2/\epsilon)$ for this scheduling problem is described in Sahni (1976, 1977). To see that this problem can be formulated as a Triangular Knapsack problem, recall that the profit is independent of the order in which the jobs are performed, and depends only on the relationship of the finish times to the due dates. The set of optimal schedules must therefore include at least one schedule in which all jobs that are completed on time are performed before those that are finished late; so the search for an optimal schedule can be restricted to schedules of this kind. Define the variables x_j , $j = 1, \dots, n$, by $x_j = 1$ if job j is completed on time, and $x_j = 0$ otherwise. The Max JSD problem can then be formulated as:

Max Job Sequencing with Due Dates (Max JSD), Version 2

Instance: (n, p, t, d, K) . The number of jobs n ; the profits p_j , $j = 1, \dots, n$; the execution times t_j , $j = 1, \dots, n$; the due dates d_j , $j = 1, \dots, n$; and the minimum profit target K ; are non-negative integers such that $t_j \leq d_j$, $j = 1, \dots, n$, and $d_j \leq d_{j+1}$, $j = 1, \dots, n - 1$.

Question: Is there an x such that

$$\begin{aligned} \sum_{j=1}^n p_j x_j &\geq K \\ \sum_{k=1}^j t_k x_k &\leq d_j, \quad j = 1, \dots, n \\ x_j &\in \{0, 1\}, \quad j = 1, \dots, n \end{aligned}$$

This has the form of a Triangular Knapsack problem. The results from the previous section imply the existence of an FPTAS for the multi-criteria form of this problem.

6.3 The Minimum Job Sequencing with Due Dates Problem

A different version of the Job sequencing with Due Dates problem is the one where jobs are assessed penalties if they finish late. In this case the objective is to minimize the sum of the penalties and the

problem is called the Min Job Sequencing with Due Dates (Min JSD) problem. An $O(n^2 \log n + n^2/\epsilon)$ FPTAS for this problem is described by Gens and Levner (1979).

The Min JSD problem is not just a trivial variant of the Max JSD problem, because the complexities of approximating the solution of the minimization problem and the corresponding maximization problem can be quite different. Nevertheless, the approximation schemes of Gens and Levner (1979) are derived by reducing the Min JSD problem to the Min Arborescent Knapsack problem. The variables are the complements of those used in the Max JSD problem, i.e., $y_j = 1$ if job j is completed late, and 0 otherwise. The problem can be formulated as follows:

Min Job Sequencing with Due Dates (Min JSD), Version 1

Instance: (n, p, t, d, K) . The number of jobs n ; the penalties $p_j, j = 1, \dots, n$; the execution times $t_j, j = 1, \dots, n$; the due dates $d_j, j = 1, \dots, n$; and the maximum penalty target K ; are non-negative integers such that $t_j \leq d_j, j = 1, \dots, n$ and $d_j \leq d_{j+1}, j = 1, \dots, n - 1$.

Question: Is there a y such that

$$\begin{aligned} \sum_{j=1}^n p_j y_j &\leq K \\ \sum_{k=1}^j t_k (1 - y_k) &\leq d_j, \quad j = 1, \dots, n \\ y_j &\in \{0, 1\}, \quad j = 1, \dots, n, \end{aligned}$$

which is a Triangular Knapsack problem. Setting $d'_j = \sum_{k=1}^j t_k - d_j$ allows the Min JSD problem to be formulated as a knapsack covering problem:

Min Job Sequencing with Due Dates (Min JSD), Version 2

Instance: (n, p, t, d, K) . The number of jobs n ; the penalties $p_j, j = 1, \dots, n$; the execution times $t_j, j = 1, \dots, n$; the due dates $d_j, j = 1, \dots, n$; and the maximum penalty target K ; are non-negative integers such that $t_j \leq d_j, j = 1, \dots, n$, and $d_j \leq d_{j+1}, j = 1, \dots, n - 1$.

Question: Is there a y such that

$$\begin{aligned} \sum_{j=1}^n p_j y_j &\leq K \\ \sum_{k=1}^j t_k y_k &\geq d'_j, \quad j = 1, \dots, n \\ y_j &\in \{0, 1\}, \quad j = 1, \dots, n, \end{aligned}$$

This is a Binary Aborescent Knapsack Covering problem. The previously known results for the Min JSD problem can therefore be generalized to guarantee an FPTAS for the Integer Aborescent Knapsack Covering problem with multiple criteria.

6.4 The Partially-Ordered Knapsack Problem

Another special case of the Multi-Dimensional Knapsack problem whose structure allows for the existence of an FPTAS is the Partially-Ordered Knapsack problem. In this variant, items must be chosen in accordance with specified precedence constraints. Such a constraint requires that, for instance, if an item j is selected, then another specified item i must also be selected.

Requirements of this nature are quite common. For example, a manufacturer must decide which items to produce in fixed time period, but must produce the components for certain complex assemblies before producing the assemblies themselves.

Precedence constraints define a partial order “ \rightsquigarrow ” on the set of items; if item i must be selected whenever item j is selected, then $i \rightsquigarrow j$ (read i precedes j). A partial order \rightsquigarrow can be represented by a directed graph $G(\rightsquigarrow) = (N, A)$; N contains a node for each item, and for each $i, j \in N$, A contains arc (i, j) if and only if $i \rightsquigarrow j$. We assume that the graph $G(\rightsquigarrow)$ is acyclic and connected. Several kinds of partial orders are particularly interesting.

Definition: A partial order \rightsquigarrow is a *tree partial order* if the undirected version of $G(\rightsquigarrow)$ is a tree. A tree partial order is an *in-tree partial order* if for some “root” node in $G(\rightsquigarrow)$, all arcs point toward the root; it is an *out-tree partial order* if all arcs point away from the root.

The feasible set for the Partially-Ordered Knapsack problem is defined as follows.

Partially-Ordered Knapsack Feasible Set

Instance: $S(n, V, v, \rightsquigarrow)$. The number of items n ; the knapsack volume V ; and the item volumes $v_j, j = 1, \dots, n$; are non-negative integers such that $v_j \leq V, j = 1, \dots, n$. The partial order \rightsquigarrow is defined on the set of items $\{1, \dots, n\}$.

Feasible Set: $S(n, V, v, \rightsquigarrow)$ is the set of all $x \in Z^{n \geq}$ satisfying

$$\begin{aligned} \sum_{j=1}^n v_j x_j &\leq V \\ x_i &\geq x_j \quad i, j = 1, \dots, n, \text{ such that } i \rightsquigarrow j \\ x_j &\in \{0, 1\} \quad j = 1, \dots, n \end{aligned}$$

The Partially-Ordered Knapsack problem with a single criterion of the form $\sum_{j=1}^n p_j x_j$ is strongly NP -hard when general partial orders are allowed, but can be solved in pseudo-polynomial time when only tree partial orders are allowed (see Garey and Johnson, 1979). A PTAS for the maximization form of this single-criterion problem, restricted to in-tree partial orders, is described in Ibarra and Kim (1978); it runs in time $O(n^{2+1/\epsilon}/\epsilon)$. The algorithm of Ibarra and Kim (1978) is also applied to the minimization form of the problem in which the knapsack constraint is of the covering type and the precedence constraints form an in-tree partial order.

Johnson and Niemi (1983) contains several stronger results: an $O(n^2/\epsilon^2 + n^2 \log(P^*))$ FPTAS is presented for out-tree partial orders and an $O(n^3/\epsilon)$ FPTAS is given for the in-tree case, where P^* is the optimal value. The following theorem generalizes the results of Johnson and Niemi (1983) by allowing multi-criteria objective functions and general tree partial orders.

Theorem 8: Let Ψ be the collection of feasible regions for the Partially-Ordered Knapsack problem with general tree partial orders. Then the problem $(\Psi, \mathcal{AM}(r), Opt)$, for any $r \in Z^+$, has an FPTAS.

Proof: The standard dynamic programming approach can be used to solve the STT Network Flow problem $(\Psi, \mathcal{AP}_1, Opt)$ in VPP time; see Johnson and Niemi (1983) for details. The rest follows by using VPP reduction. \square

Reversing the constraints that define the feasible region produces a Partially-Ordered Knapsack Covering problem with precedence constraints the reverse of those in the original problem. Note that if the original precedence constraints form a tree partial order, then the reversed constraints also form a tree partial order. If the original precedence constraints are reversed before the covering transformation is applied, then a covering problem with the original precedence constraints results. The following conclusion is therefore valid, even though not all the original constraints are reversed to pose the covering form of a Partially-Ordered Knapsack problem.

Corollary 4: The Partially-Ordered Knapsack Covering problem with general tree partial orders has an FPTAS.

7 Production Planning Problems

The production planning problem considered here is a capacitated, single-commodity, dynamic lot sizing problem. That is, a facility manufacturing a single product has to satisfy known integer demands over a planning horizon of n periods. Demand may be backlogged and time-dependent manufacturing capacities are specified. The object is to determine how many units, if any, to produce each period in order to minimize the (possibly non-linear) costs of production and inventory storage over a finite horizon.

Given demands d_j , production capacities c_j , and production quantities x_j , define the following quantities at the end of each period j : $D_j = \sum_{i=1}^j d_i$ (cumulative demand), $C_j = \sum_{i=1}^j c_i$ (cumulative capacity), $X_j = \sum_{i=1}^j x_i$ (cumulative production), and $I_j = X_j - D_j$ (inventory at the end of period j). The feasible set for this problem is defined as follows:

Capacitated Single Commodity Dynamic Lot Sizing Feasible Set

Instance: (n, d, c, s, t) . The number of periods n ; the demands $d_j, j = 1, \dots, n$; the production capacities $c_j, j = 1, \dots, n$; the inventory capacities $s_j, j = 1, \dots, n$; and the backordering capacities $t_j, j = 1, \dots, n$; are non-negative integer such that $D_j \leq C_j, j = 1, \dots, n$.

Feasible Set: $S(n, d, c)$ is the set of all $x \in Z^{n \geq}$ satisfying

$$\begin{aligned} x_j &\leq c_j, & j = 1, \dots, n \\ I_j &\leq s_j, & j = 1, \dots, n \\ I_j &\geq -t_j, & j = 1, \dots, n \\ I_0 &= 0 \\ I_n &= 0 \end{aligned}$$

The assumption that $D_j \leq C_j$ is necessary and sufficient for the feasible set to be nonempty (Florian and Klein, 1971). The constraints $I_0 = 0$ and $I_n = 0$ are convenient but can be removed without loss of generality.

The cost functions typically used with this problem are:

$$\begin{aligned} p_j(x_j) &\quad \text{cost of producing } x_j \text{ units in period } j, \\ h_j(I_j) &\quad \text{cost of holding } I_j \text{ units in inventory at the end of period } j, \text{ and} \\ b_j(I_j) &\quad \text{cost of backordering } I_j \text{ units in period } j. \end{aligned}$$

This is an important problem that has received a lot of attention in the literature. Florian et al. (1980) and Bitran and Yanasse (1982) provide snapshots of the state-of-the-art with regard to known complexity results for variants of this problem. Linear production and holding costs and time-dependent setup costs are handled by an algorithm in Baker et al. (1978); it runs in time $O(n^3 D_n + n D_n C_n)$. A modification that allows time-dependent capacities and backloging and runs in time $O(n^4 D_n + n^2 D_n C_n)$ is described in Florian et al. (1980). The performance of various heuristics is studied in Bitran et al. (1984) and Bitran and Matsuo (1986), and an FPTAS for a continuous-time problem appears in Gavish and Johnson (1990).

The time bounds for previous algorithms for the problem considered here, although pseudo-polynomial, are not VPP. This means that none of them can be directly converted into an FPTAS in the manner described earlier, with the exceptions of the FPTAS presented in Dada and Orlin (1981). More recently, van Hoesel and Wagelman (2001) have presented a new FPTAS for this problem.

In the theorem below, we show that the multi-criteria Capacitated Single-Commodity Dynamic Lot Sizing problem has an FPTAS. It therefore also has a VPP algorithm which, for some situations, may be preferable to the pseudo-polynomial time algorithms already known.

Theorem 9: Let Ψ be the collection of Capacitated Single-Commodity Dynamic Lot Sizing feasible sets. The problem $(\Psi, \mathcal{AM}_r, Opt)$, for any $r \in Z^+$, has an FPTAS.

Proof: Let $\Pi_1 = (\Psi, \mathcal{AM}_r, Feas)$ and $\Pi_2 = (C_2, \mathcal{AM}_r, Feas)$, where the feasible set C_2 is an STT network with no parallel arcs in the tree part of the network, general flow capacities, unit flow multipliers, only demand nodes, and exact flow requirements. We show a reduction $\Pi_1 \propto_{VPP} \Pi_2$. The FPTAS result follows from Lemma A4 and Theorem 5.

Suppose that an instance $I_1 = (S, f, M)$ of Π_1 is given, with $S = S(n, d, c)$. The objective function f is the sum over the decision time periods of the functions p_j, h_j , and b_j described above. Define an instance $I_2 = (C_2, f, M)$ of Π_2 , where C_2 is defined by an STT network $G = (N, A, u, \mu, b)$. The set N has $(n + 1)$ nodes: a node β_j corresponding to each period j , with demand d_j ; and node 0. The set A has $(3n - 2)$ arcs that are described in the following table.

Arc	Capacity	Multiplier	Value
$(0, \beta_j), \quad j = 1, \dots, n$	c_j	1	$p_j(x_{0,j})$
$(\beta_j, \beta_{j+1}), \quad j = 1, \dots, n - 1$	s_j	1	$h_j(x_{j,j+1})$
$(\beta_{j+1}, \beta_j), \quad i = 1, \dots, n - 1$	t_j	1	$b_j(x_{j+1,j})$

The production in period j corresponds to the value $x_{0,j}$, and the cost of a production schedule is the same as the cost of the corresponding flow in C_2 . The reduction associates a solution of I_1 with a solution of I_2 that has the same objective function value. \square

8 Summary

This work has introduced a new framework in which to consider the solution of multi-criteria combinatorial optimization problems. Since finding the entire efficient frontier can be difficult, the approach taken here has been to approximate it.

Necessary and sufficient conditions for the existence of fully polynomial time approximation schemes were motivated and proved. These conditions, when restricted to the single criterion case, are more specific than the previously known results. The results necessitated the specification of the regularity conditions of closure under scaling and closure under box constraints. Besides being technically necessary for the proofs, the former condition is a real concern in practice and the latter does not sacrifice any substantive generality. These conditions were generalized to the quasi-closure form so that additively separable objective functions can be handled.

Next, the STT Network Flow problem was introduced; this problem generalizes a wide variety of commonly-studied combinatorial optimization problems. A VPP algorithm for solving a simple version of this problem was demonstrated and then used to show that the problem has an FPTAS. The boundary between easy and hard problems on STT networks was then explored in some detail.

The computational complexity of finding approximate solutions to several problems that arise frequently in applications was also studied. For instance, the previously known FPTAS for the binary case of a single-criterion Aborescent Knapsack problem was generalized to the integer case with multiple criteria. An FPTAS for the covering form of this problem was also demonstrated. Machine scheduling problems such as Max Job Sequencing with Deadlines and Min Job Sequencing with Deadlines were cast in this framework, with FPTAS approaches to the multi-criteria cases. An FPTAS for the multi-criteria form of the Partially-Ordered Knapsack problem with general tree constraints was also shown, generalizing the previously known single-criterion results. A covering form of this problem was also shown to have FPTAS. Finally, The Capacitated Single Commodity Dynamic Lot Sizing problem was considered. By reducing this problem to an STT Network Flow problem, the existence of an FPTAS was demonstrated.

9 Bibliography

Ahuja, R.K., T.L. Magnanti, and J.B. Orlin, (1993). *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Baker, K.R., P. Dixon, M.J. Magazine, and E.A. Silver, (1978). "An algorithm for the dynamic lot-sizing problem with time-varying production capacity constraints", *Management Science* 24, 1710-1720.

Bitran, G.R. and H.H. Yanasse, (1982). "Computational complexity of the capacitated lot size problem", *Management Science* 28, 1174-1186.

Bitran, G.R., T.L. Magnanti, and H.H. Yanasse, (1984). "Approximation methods for the uncapacitated dynamic lot size problem", *Management Science* 30, 1121-1140.

Bitran, G.R. and Matsuo, H., (1986). "Approximation functions for the capacitated lot size problem", *Operations Research* 34, 63-74.

Dada, M. and J.B. Orlin, (1981). "The capacitated multi-item dynamic lot-size problem: Exact and ϵ -optimal algorithms", Presented at Spring CORS/TIMS/ORSA Conference, Toronto, 1981.

Ecker, J.G., and I.A. Kouada, (1975). "Finding efficient points for linear multiple objective programs", *Mathematical Programming* 8, pp. 375-377.

Florian, M. and M. Klein, (1971). "Deterministic production planning with concave costs and capacity constraints", *Management Science* 18, 12-20.

Florian, M., J.K. Lenstra, and A.H.G. Rinnooy Kan, (1980). "Deterministic production planning: Algorithms and Complexity", *Management Science* 26,669-679.

Ford, L.R.,Jr. and D.R. Fulkerson (1962). *Flows in Networks*, Princeton University Press.

Garey, M.R. and D.S. Johnson, (1978). "'Strong' NP-completeness results: Motivation, examples, and implications", *J. ACM* 25, 499-508.

Garey, M.R. and D.S. Johnson, (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman & Co., San Francisco, California.

Gavish, B. and R.E. Johnson, (1990). "A fully polynomial approximation scheme for single product scheduling in a finite capacity facility", *Operations Research* 38, 70-83.

Gens, G.V. and E.V. Levner, (1979). "Computational complexity of approximation algorithms for combinatorial problems", In J. Becvar, (ed.), *Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science - 74, Springer-Verlag, New York, 292-300.

Graham, R.J., E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, (1979). "Optimization and approximation in deterministic sequencing and scheduling: A survey", In P.L. Hammer, E.L. Johnson, and B.H. Korte, editors, *Discrete Optimization II*, 287-326, North-Holland Publishing Co., New York, 1979.

Hochbaum, D.S., (Eds.) (1997). *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Company, Boston, Massachusetts.

Ibarra, O.H. and C.E. Kim, (1978). "Approximation algorithms for certain scheduling problems", *Mathematics of Operations Research* 3, 197-204.

Ibarra, O.H. and C.E. Kim, (1975). "Fast approximation algorithms for the knapsack and sum of subsets problems", *J. ACM* 22, 197-204.

Johnson, D.S. and K.A. Niemi (1983). "On knapsacks, partitions, and a new dynamic programming technique for trees", *Mathematics of Operations Research* 8, 1-14.

Kannan, R. and B.H. Korte, (1984). "Approximate combinatorial algorithms", in Cottle, R.W., M.L. Kelmanson, and B.H. Korte, (eds.), *Mathematical Programming*, Proceedings of the International Conference on Mathematical Programming, North-Holland Publishing Co., New York, pp. 195-248.

Knuth, D.E., *The Art of Computer Programming*, volume 1: Fundamental Algorithms. Addison-

Wesley Publishing Co., Reading, MA. second edition, 1973.

Lawler, E.L., (1979). “Fast approximation algorithms for knapsack problems”, *Mathematics of Operations Research* 4, 339-356.

Orlin, J.B., (1982). “Fast Approximation Schemes for Combinatorial Optimization Problems”, Presentation at TIMS/ORSA meeting, San Diego, California.

Papadimitriou, C.H. and K. Steiglitz, (1982). *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Papadimitriou, C.H. and M. Yannakakis, (2000). “On the approximability of trade-offs and optimal access of web sources”, Extended Abstract, *Proc. 41st IEEE Symp. on Foundations of Computer Science* 2000.

Safer, H. M. and J. B. Orlin, (1995a). “Fast Approximation Schemes for Multi-Criteria Combinatorial Optimization”, WP #37856-95, Operations Research Center, MIT, Cambridge, Massachusetts.

Safer, H. M. and J. B. Orlin, (1995b). “Fast Approximation Schemes for Multi-Criteria Flow, Knapsack, and Scheduling Problems”, WP #37855-95, Operations Research Center, MIT, Cambridge, Massachusetts.

Sahni, S., (1976). “Algorithms for scheduling independent tasks”, *J. ACM* 22, 116-127.

Sahni, S., (1977). “General Techniques for Combinatorial Optimization”, *Operations research* 25, 920-936.

Steuer, M.K., (1986). *Multiple Criteria Optimization: Theory, Computation, and Application*, John Wiley & Sons, New York, New York.

Van Hoesel, C.P.M. and A.P.M. Wagelmans, (2001). “Fully polynomial approximation schemes for single-item capacitated economic lot-sizing problems”, *Math of OR* 26, 339-357.

Vazirani, V.V., (2003). *Approximation Algorithms*, Springer-Verlag, Berlin, Germany.

Warburton, A., (1987). "Approximation of pareto optima in mulitple-objective, shortest-path problems", *Operations Research* 35, 70-79.

Woeginger, G.J., (2000). "When does a dynamic programming formulation guarantee the existance of fully polynomial time approximation scheme (FPTAS)?", *INFORMS J. on Computing* 12, 57-74.

Appendix A: VPP Reductions

When proving that a problem has an FPTAS, reduction between problems can be used to avoid having to explicitly demonstrate a VPP algorithm. This appendix introduces a new kind of reduction that is appropriate to discussions of the existence and non-existence of VPP algorithms and FPTASs.

Definition A1: A problem Π *VPP reduces* to problem Π' , written $\Pi \propto_{VPP} \Pi'$, if there is an algorithm \mathcal{A} for Π that uses as a subroutine an algorithm \mathcal{A}' for Π' , such that if \mathcal{A}' is a VPP algorithm then \mathcal{A} is a VPP algorithm.

Algorithm \mathcal{A} is called a *VPP reduction* of Π to Π' . If both $\Pi \propto_{VPP} \Pi'$ and $\Pi' \propto_{VPP} \Pi$, then Π and Π' are *VPP equivalent*, written $\Pi \equiv_{VPP} \Pi'$.

Proving that Problems are Easy Using VPP Reductions

This section shows how to use VPP reductions to clarify the difficulty of solving specific problems. VPP reduction can be used to simplify a proof that a problem has an FPTAS and it can also be used to show that a problem is difficult. The following lemmas express simple relations between problems with respect to VPP reducibility.

Lemma A1: Let Ψ be a family of feasible sets. Suppose that \mathcal{F} and \mathcal{F}' are families of r -criteria functions such that $\mathcal{F} \subseteq \mathcal{F}'$. Then $(\Psi, \mathcal{F}, Feas) \propto_{VPP} (\Psi, \mathcal{F}', Feas)$.

The next lemma shows that achieving a target value exactly is at least as difficult as finding a value between the target and the efficient frontier.

Lemma A2: Let Ψ be a family of feasible sets and \mathcal{F} a family of r -criteria functions that is quasi-closed under box constraints. Let $\Pi = (\Psi, \mathcal{F}, Feas)$ be a feasibility problem and $\Pi^=$ be the same problem in which the target value must be achieved exactly. Then $\Pi \propto_{VPP} \Pi^=$.

Proof: Given $I = (S, f, M) \in \Pi$, let $g \in \Psi$ be a box constraint neighbor of f with respect to M . So for all $x \in S$, $\|g(x)\| \leq nM$.

Define $H = \{M' \in Z^{r+} : M' \leq nM\}$. If I has a solution $x \in S$, then $g(x) \in H$. Since $|H| = (n \times \|M\| + 1)^r$, a VPP reduction consists of solving the instance $I^\# = (S, g, M') \in \Pi^\#$ for each $M' \in H$ until a solution x is found that also solves I . \square

A feasibility problem is sometimes easier to work with than an optimization problem, as the former can be solved by finding a single point. Proofs about optimization problems can often be simplified by using the following lemma.

Lemma A3: Let Ψ be a family of feasible sets and \mathcal{F} a family of r -criteria functions that is quasi-closed under box constraints. Then $(\Psi, \mathcal{F}, Feas) \equiv_{VPP} (\Psi, \mathcal{F}, Opt)$.

Proof: Let $\Pi_F = (\Psi, \mathcal{F}, Feas)$ and $\Pi_O = (\Psi, \mathcal{F}, Opt)$.

$\Pi_F \propto_{VPP} \Pi_O$: Let $I_F = (S, f, M) \in \Pi_F$ and $g \in \Psi$ be a box constraint neighbor of f with respect to M ; so $I_O = (S, g) \in \Pi_O$. A solution set C_O for I_O contains at most $(n \times \|M\| + 1)^r$ points. At least one of these points solves I_F , if I_F has a solution. So a VPP reduction consists of solving I_O and examining members of C_O until a solution to I_F is found.

$\Pi_O \propto_{VPP} \Pi_F$: Let $I_O = (S, f) \in \Pi_O$, and define the set of target vectors

$$H = \{M \in Z^{r+} : \|M\| \leq \bar{M}_v(I_O)\};$$

so $|H| = (\bar{M}_v(I_O) + 1)^r$. For each target vector $M \in H$, let $C_F(M)$ be the solution to $(S, f, M) \in \Pi_F$, and define $C_F = \cup_{M \in H} C_F(M)$. The solution to I_O is a subset of C_F . A VPP reduction consists of solving (S, f, M) for each $M \in H$ and eliminating unneeded elements from C_F . \square

Corollary A1: Let Ψ be a family of feasible sets and \mathcal{F} and \mathcal{F}' families of r -criteria functions that are quasi-closed under box constraints and satisfy $\mathcal{F} \subseteq \mathcal{F}'$. Let $\Pi = (\Psi, \mathcal{F}, Opt)$, $\Pi' = (\Psi, \mathcal{F}', Opt)$, and $\Pi^\# = \Pi$, but in which we want to find all possible function values. Then $\Pi \propto_{VPP} \Pi'$ and $\Pi \propto_{VPP} \Pi^\#$.

This leads to the following useful lemma.

Lemma A4: Let Ψ be a family of feasible sets and \mathcal{F} a family of r -criteria functions that is quasi-closed under scaling and under box constraints. Let Π' be a feasibility or optimization problem that

has a VPP algorithm or an optimization problem that has an FPTAS. If $(\Psi, \mathcal{F}, Feas) \propto_{VPP} \Pi'$ or $(\Psi, \mathcal{F}, Opt) \propto_{VPP} \Pi'$, then (Ψ, \mathcal{F}, Opt) has an FPTAS.

Proof: If Π' is an optimization problem that has an FPTAS, then by Theorem 1 it has a VPP algorithm. So in all cases, Π' has a VPP algorithm.

If $(\Psi, \mathcal{F}, Feas) \propto_{VPP} \Pi'$, then by Lemma A3, $(\Psi, \mathcal{F}, Opt) \propto_{VPP} \Pi'$ as well; so under either reduction assumption, (Ψ, \mathcal{F}, Opt) has a VPP algorithm. Theorem 2, then, implies that (Ψ, \mathcal{F}, Opt) has an FPTAS. \square

Proving that Problems are Hard Using VPP Reductions

This section examines the other side of the coin and illustrates one way to use VPP reductions to prove that problems are difficult. The discussion begins with a general setting; then a specific parameter choice is made in order to prove that certain problems cannot have VPP algorithms. For a different approach and discussion of strong *NP*-hardness see Garey and Johnson (1979).

Let β be a set of integers of interest for instances of a problem Π . In the present setting, an appropriate choice for β will be seen to be V , the set of possible objective function values.

Definition A2: Let $p(\cdot)$ be a single-variable polynomial. The problem $\Pi_{\beta,p}$ is *the restriction of the problem Π to those instances in which the integers in β are bounded by the function $p(\cdot)$ of the instance length*, that is, $\Pi_{\beta,p} = \{I \in \Pi : \forall b \in \beta, b \leq p(L(I))\}$.

Definition A3: A problem Π is *β -strongly *NP*-hard* if, for some polynomial $p(\cdot)$, the problem $\Pi_{\beta,p}$ is *NP-hard*.

This definition generalizes a variety of past work. If β is the set of all the integers that appear in the instance, then β -strong *NP*-hardness is the same as strong *NP*-hardness. On the other hand, if $\beta = \emptyset$, then β -strong *NP*-hardness is the same as ordinary *NP*-hardness.

Recall that if $P \neq NP$, then no *NP*-hard problem has a polynomial time algorithm, and no strongly *NP*-hard problem has a pseudo-polynomial time algorithm (Garey and Johnson, 1978). In the same vein, the following theorem shows that the appropriate choice for the present context

is $\beta = V$.

Theorem A1: If a problem Π is V -strongly NP -hard, then Π has no VPP algorithm unless $P = NP$.

Proof: Since Π is V -strongly NP -hard, there is a polynomial $p(\cdot)$ such that $\Pi_{V,p}$ is NP -hard. Suppose that Π can be solved by a VPP algorithm \mathcal{A} ; then \mathcal{A} solves instances $I \in \Pi_{V,p}$ in polynomial time, since $M_v(I) \in O([L(I)]^k)$, for some $k \in \mathbb{Z}^+$. But $\Pi_{V,p}$ is NP -hard, so such an algorithm \mathcal{A} can exist only if $P = NP$. \square

Once a problem has been shown to be V -strongly NP -hard, VPP reductions can be used to prove that other problems have this property. We state this formally in the next theorem. First, the lemma below shows that the magnitudes of the values that occur in constructed instances are not too large.

Lemma A5: Let Π and Π' be problems such that $\Pi \propto_{VPP} \Pi'$, and let \mathcal{A} be a reduction algorithm that calls an algorithm \mathcal{A}' for Π' . Suppose that while \mathcal{A} is solving some $I \in \Pi$, it calls \mathcal{A}' to solve some $I' \in \Pi'$. The the largest value of I' is polynomial in both the length and the largest value of I , that is, $M_v(I') \in O([L(I) \times M_v(I)]^k)$, for some $k \in \mathbb{Z}^+$.

Proof: By contradiction. \square

Theorem A2: Let Π and Π' be two problems. If Π is V -strongly NP -hard and $\Pi \propto_{VPP} \Pi'$, then Π' is also V -strongly NP -hard.

Proof: Let \mathcal{A} be a VPP reduction algorithm for solving Π that calls an algorithm \mathcal{A}' for Π' . Since Π is V -strongly NP -hard, there is a polynomial $p_1(\cdot)$ such that Π_{V,p_1} is NP -hard. Let $I \in \Pi_{V,p_1}$, and suppose that while \mathcal{A} is solving I , it calls \mathcal{A}' to solve some $I' \in \Pi'$. Then there are polynomials $p_2(\cdot)$ and $p_3(\cdot)$ such that

$$M_v(I') \leq p_2(L(I) \times M_v(I)) \leq p_2(L(I) \times p_1(L(I))) \leq p_3(L(I))$$

Assume for the moment that $L(I) \in O([L(I')]^k)$, for some $k \in \mathbb{Z}^+$. Then for some polynomial $p_4(\cdot)$, $M_v(I') \leq p_4(L(I'))$. This holds for all I' for which \mathcal{A}' is called, so $\Pi_{V,p_1} \propto_{VPP} \Pi'_{V,p_4}$.

Since $M_v(I) \leq p_1(L(I))$, however, the algorithm \mathcal{A} runs in time polynomial in $L(I)$; it is therefore also the case that $\Pi_{V,p_1} \propto_T \Pi'_{V,p_4}$, where \propto_T denotes polynomial time Turing reducibility (see Garey and Johnson, 1979, section 5.1). But Π_{V,p_1} is *NP*-hard, so Π'_{V,p_4} is also *NP*-hard, and hence Π' is *V*-strongly *NP*-hard.

The proof depends on the assumption that $L(I) \in O([L(I')]^k)$, for some $k \in \mathbb{Z}^+$; this means that the length of the constructed instance I' is not much shorter than the length of the original instance I . The description of I' can be padded with enough blanks so that this is true, so the assumption can be made without loss of generality. \square