Fast Approximation Schemes for Multi-Criteria Combinatorial Optimization

by
Hershel M. Safer
James B. Orlin

WP #3756-95 January 1995

Fast Approximation Schemes For Multi-Criteria Combinatorial Optimization

Hershel M. Safer
Genome Therapeutics Corp.
h.safer@ieee.org

James B. Orlin
Sloan School of Management
Massachusetts Institute of Technology
jorlin@mit.edu

4 January 1995

Fast Approximation Schemes For Multi-Criteria Combinatorial Optimization

Hershel M. Safer and James B. Orlin

Abstract

The solution to an instance of the standard Shortest Path problem is a single shortest route in a directed graph. Suppose, however, that each arc has *both* a distance *and* a cost, and that one would like to find a route that is *both* short *and* inexpensive. In general, no single route will be both shortest and cheapest; rather, the solution to an instance of this multi-criteria problem will be a set of efficient or Pareto optimal routes. The (distance, cost) pairs associated with the efficient routes define an efficient frontier or tradeoff curve.

An efficient set for a multi-criteria problem can be exponentially large, even when the underlying single-criterion problem is in $\mathcal{P}$. This work therefore considers *approximate* solutions to multi-criteria discrete optimization problems and investigates when they can be found quickly. This requires generalizing the notion of a fully polynomial time approximation scheme to multi-criteria problems.

In this paper, necessary and sufficient conditions are developed for the existence of such a *fast approximation scheme* for a problem. Although the focus is multi-criteria problems, the conditions are of interest even in the single criterion case. In addition, an appropriate form of problem reduction is introduced to facilitate the application of these conditions to a variety of problems. A companion paper uses the results of this paper to study the existence of fast approximation schemes for several interesting network flow, knapsack, and lot-sizing problems.

Contents

List of Figures

Glossary

This section contains brief explanations of many symbols and terms that are used in this paper and its companion paper, along with references to their full definitions.

Symbols

$\mathcal{AP}(\omega)$, $\mathcal{AR}(\omega)$, $\mathcal{AM}(\omega)$	The set of additively separable functions that are, respectively, in $\mathcal{P}(\omega)$, $\mathcal{R}(\omega)$, and $\mathcal{M}(\omega)$. ([2], section 3.2)
$\mathcal{C}$	$(A1/CG/MG/NM/FS)$ ([2], section 2)
$\mathcal{C}'$	The subclass of $\mathcal{C}$ consisting of those networks in canonical form. ([2], section 4.1)
$D_\omega(u,v)$	u dominates v w.r.t. ω. ([1], section 3.3)
$D_\omega^{(\varepsilon)}(u,v)$	u ε-dominates v w.r.t. ω. ([1], section 3.4)
e_k	A vector of length k in which each component is one. ([1], section 3)
$e_v(x)$	The excess flow into node v that arises from the flow x. ([2], section 2)
ε	A non-negative bound on relative error. ([1], section 3.4)
$\equiv_{\text{VPP}}$	VPP equivalent. ([1], section 5.1)
f_i	The i^{th} component of the multi-criteria objective function f. ([1], section 3.1)
$\mathcal{F}$	A family of r-criteria objective functions. ([1], section 3.1)
$\lfloor f/v \rfloor$	The function f scaled by v. ([1], section 4.2)
$G(\rightsquigarrow)$	The directed graph defined by the partial order $\rightsquigarrow$. ([2], section 8)
I	An instance of a problem. ([1], section 3.1) (S, f, ω) An optimization instance. (S, f, ω, M) A feasibility instance.
$L(I)$	The length of instance I. ([1], section 3.2)
$\log(x)$	Base 2 logarithm of x. ([1], section 3)
$\min\{f,\ M\}$	The function f with box constraints M. ([1], section 4.2)
$M_V(I)$	The largest value of instance I. ([1], section 3.2)
$\overline{M}_V(I)$	A close upper bound on $M_V(I)$ that can be found quickly. ([1], section 3.2)
n	The number of variables in a problem instance. ([1], section 3.1)
$\|\|x\|\|$	The infinity norm of the vector x. ([1], section 3)
ω	A direction vector. ([1], section 3.1)
ω^D	The dual of the direction vector ω, obtained by reversing the direction of each component of ω. ([2], section 3.3)
Ω	The set of all direction vectors. ([1], section 3.1)
Ω^r	The set of all direction vectors of dimension r. ([1], section 3.1)
$\mathcal{P}(\omega)$, $\mathcal{R}(\omega)$, $\mathcal{M}(\omega)$	The set of functions that are, respectively, order-preserving, order-reversing, and order-monotone w.r.t. ω. ([2], section 3.1)
Π	A problem. ([1], section 3.1) $(\Psi, \mathcal{F}, \omega, \text{Opt})$ An optimization problem. $(\Psi, \mathcal{F}, \omega, \text{Feas})$ A feasibility problem.
$\Pi_{\beta,p}$	The restriction of problem Π to instances in which the integers in β are bounded by the polynomial function $p(\cdot)$ of the instance length. ([1], section 5.3)

r	The number of criteria in a problem. ([1], section 3.1)
$\propto_{\text{VPP}}$	VPP reduces. ([1], section 5.1)
Ψ	A family of feasible sets. ([1], section 3.1)
S	The feasible set of a problem instance. ([1], section 3.1)
S^{cov}, Ψ^{cov}	The covering forms, respectively, of the feasible set S and the family Ψ of feasible sets. ([2], section 6)
$(\cdot/\cdot/\cdot/\cdot/\cdot)$	Specification of a class of STT networks. ([2], section 1)
$u(S)$	A vector of upper bounds on the components of points in the feasible set S. ([1], section 3.1)
$u_{max}(S)$	The largest component of $u(S)$. ([1], section 3.1)
$\mathcal{Z}^+$	The set of non-negative integers. ([1], section 3.1)

In an STT network $G = (N, A, c, m, b)$: ([2], section 2)

N	The set of nodes.
A	The set of arcs.
c	The vector of arc capacities.
m	The vector of arc multipliers.
b	The vector of node demands.

In the construction of the first VPP algorithm: ([2], section 4.1)

T	The tree corresponding to the graph G.
ρ	The root of the tree T.
$l(v)$	The postorder label of node v.
d_v	The number of children of node v in T.
$T[v,j]$	The subtree of T defined by v, the first j children of v, and all the descendants of those children.
$G[v,j]$	The subgraph of G defined by node 0 and the nodes of G that correspond to the nodes of $T[v,j]$.
$\theta(v,j,k)$	The maximum excess flow into node v over all flows x that are feasible w.r.t. $G[v,j]$ and have $f(x) = k$.

Terms

Additively separable Describes a function $f : S \to \mathcal{Z}^{r+}$, where $S \subseteq \mathcal{Z}^{n+}$, that can be written as $f(x) \equiv \sum_{j=1}^{n} f_j(x_j)$. ([2], section 3.2)

Arborescent Describes a collection of sets, any two of which either are disjoint or for which one is properly contained in the other. ([2], section 7.1)

β-strongly NP-hard Describes a problem Π for which, for some polynomial $p(\cdot)$, the problem $\Pi_{\beta,p}$ is $\mathcal{NP}$-hard. ([1], section 5.3)

Binary family A family of feasible sets with domain $\{0,1\}$. ([1], section 3.1)

Box constraint Upper bound imposed on the value of a function. ([1], section 4.2)

Box constraint neighbor A function that approximates the box-constrained value of another function. ([1], section 4.2)

Closed under box constraints Describes a family of r-criteria functions for which applying box constraints to any function in the family yields another function in the family. ([1], section 4.2)

Closed under scaling Describes a family of r-criteria functions for which scaling any function in the family yields another function in the family. ([1], section 4.2)

Domain With reference to a family Ψ of feasible sets, the set of integers from zero through largest value of $u_{max}(S)$ over sets $S \in \Psi$. ([1], section 3.1)

Dominate A vector u *dominates* v w.r.t. ω if u is component-by-component better than v in the direction specified by ω. Written $D_\omega(u, v)$. ([1], section 3.3)

Efficient Set A set of points whose values define the efficient frontier. ([1], section 3.3)

ε-dominate A vector u ε-dominates v w.r.t. ω if each component of u is no more than a factor of ε worse than the corresponding component of v in the direction specified by ω. Written $D_\omega^{(\varepsilon)}(u, v)$. ([1], section 3.4)

ε-efficient set A set of points whose values are within a factor of ε of each point on the efficient frontier. ([1], section 3.4)

ε-efficient solution A minimal ε-efficient set for an instance. ([1], section 3.4)

Exact solution For an optimization instance, a minimum cardinality set of feasible points whose values constitute the efficient frontier; for a feasibility instance, a point whose value achieves the target. ([1], section 3.3)

FAS Fast approximation scheme. Describes an algorithm for a problem Π which for any $\varepsilon > 0$ and any instance $I \in \Pi$, finds an ε-efficient solution for I in time $O\left((L(I)/\varepsilon)^k\right)$, for some $k > 0$. ([1], section 3.4)

Feasible flow A flow that satisfies the flow requirements and capacity constraints. ([2], section 2)

Feasible set A bounded set $S \subseteq \mathcal{Z}^{n+}$. ([1], section 3.1)

In-tree partial order A tree partial order $\rightsquigarrow$ for which $G(\rightsquigarrow)$ contains a node toward which all the arcs of $G(\rightsquigarrow)$ point. ([2], section 8)

Largest value of an instance For an optimization instance, the largest component of the objective function on a particular box-constrained superset of the feasible region; for a feasibility instance, the largest component of the upper bound. Written $M_V(I)$. ([1], section 3.2)

Length of an instance The number of bits needed to represent the feasible set and the largest value of the instance. Written $L(I)$. ([1], section 3.2)

Objective function A function $f : S \longrightarrow \mathcal{Z}^{r+}$. ([1], section 3.1)

Order-preserving, order-reversing, order-monotone Describes a multi-criteria objective function for which increasing some components of its argument yields changes in values that are consistent with a specified direction vector. ([2], section 3.1)

Out-tree partial order A tree partial order $\rightsquigarrow$ for which $G(\rightsquigarrow)$ contains a node away from which all the arcs of $G(\rightsquigarrow)$ point. ([2], section 8)

Problem A collection of instances of the same kind (i.e., optimization or feasibility) with identical direction vectors, feasible sets from the same families, and objective functions from the same families. ([1], section 3.1)

Quasi-closure under box constraints A weaker variant of closure under box constraints. ([1], section 4.2)

Quasi-closure under scaling A weaker variant of closure under scaling. ([1], section 4.2)

Scaling Truncating the low-order bits of a function value. ([1], section 4.2)

Scaling neighbor A function that approximates the scaled value of another function. ([1], section 4.2)

STT Source-to-Tree. ([2], section 2)

STT network A generalized network which has a particular forest-like structure. ([2], section 2)

Sufficient, deficient, exact flow For a node, refers to the excess flow being, respectively, non-negative, non-positive, or zero; for a network, refers to the flow into each node having the corresponding property. ([2], section 2)

Tree partial order A partial order $\rightsquigarrow$ for which the undirected version of $G(\rightsquigarrow)$ is a tree. ([2], section 8)

VPP V-pseudo-polynomial. Describes an algorithm for a problem Π which for any instance $I \in \Pi$, finds a solution for I in time $O\left(\left(L\left(I\right) M_{\mathrm{V}}\left(I\right)\right)^{k}\right)$, for some $k > 0$. ([1], section 3.3)

VPP equivalent Describes two problems, each of which is VPP reducible to the other. ([1], section 5.1)

VPP reduction A VPP algorithm for a problem that uses, as a subroutine, a VPP algorithm for another problem. ([1], section 5.1)

w.r.t. With respect to. ([1], section 3.3)

References for Glossary

[1] Hershel M. Safer and James B. Orlin, *Fast Approximation Schemes For Multi-Criteria Combinatorial Optimization*, 1994.

[2] Hershel M. Safer and James B. Orlin, *Fast Approximation Schemes For Multi-Criteria Flow, Knapsack, and Scheduling Problems*, 1994.

1 Introduction

The solution to an instance of the standard Shortest Path problem is a single shortest route in a directed graph. Suppose, however, that each arc has *both* a distance *and* a cost, and that a route that is *both* short *and* inexpensive is to be found. Unfortunately, a typical instance of this *multi-criteria* problem does not contain a route that is optimal for both criteria. Instead, a solution generally consists of multiple *efficient* or *Pareto optimal* routes, each with the characteristic that the value of one objective can be improved within the feasible region only at the expense of the value of the other objective. The objective function values of these points define a *tradeoff curve* or *efficient frontier*.

Finding a set of efficient points is often difficult, in part because such a set can be exponentially large. Furthermore, just determining if a particular feasible point is efficient is $\mathcal{NP}$-hard for many interesting problems, even when the underlying single-criterion problem is in $\mathcal{P}$.

This paper examines the possibility of solving multi-criteria problems approximately instead of exactly. This idea is not new; methods for finding approximate solutions have been discussed extensively for single-criterion optimization problems. In the single-criterion context, an *ε-approximate algorithm* finds a solution whose relative error from optimal 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 $\mathcal{P} = \mathcal{NP}$.

For some other problems, however, ε-approximate algorithms exist for *any* positive value of ε. An *approximation scheme* for a problem finds an ε-approximate solution for any specified $\varepsilon > 0$. Two classes of approximation schemes have received extensive attention. For any fixed $\varepsilon > 0$, 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 $1/\varepsilon$. An FPTAS is sometimes called a *fast approximation scheme* (FAS)[GL79a, GL79b].

In contrast to the literature on single-criterion problems, most previous work on discrete multi-criteria optimization problems has focused on exact solutions. When the efficient frontier contains many points, however, exact solution techniques fail to terminate in a reasonable amount of time. Partial solutions may be generated, but without any guarantee that the portion that has been identified is better than the part that remains unknown.

The approach described here, however, generates solutions that characterize the *entire* efficient frontier by extending the notion of FAS to multi-criteria problems. Although this idea has been used for some specific problems, it has not been discussed previously in the context of general combinatorial optimization.

The primary results of this paper are necessary and sufficient conditions for the existence of an FAS for a multi-criteria discrete optimization problem. The existence of a pseudo-polynomial time algorithm to solve a problem is known to be necessary for the problem to have an FAS[GJ78]. It is also known that this is not sufficient, but that the existence of an algorithm from a particular subclass of the pseudo-polynomial time algorithms is sufficient[PS82]. Besides extending these conditions to the multi-criteria case, this paper bridges the gap between the necessary and sufficient conditions by identifying a smaller subclass that is appropriate for both the necessary and sufficient conditions. The conditions derived here are therefore of interest even in the single-criterion case. In addition, a new form of reduction that can be used to simplify the application of the main theorem is introduced.

Section 1.1 highlights the foundations of this work, and Section 2 discusses the motivation for examining the questions addressed here. Terminology and symbols are introduced in Section 3. The statement and proof of the necessary and sufficient conditions for the existence of an FAS are presented in Section 4, and problem reductions are described in Section 5.

A companion paper[SO94] applies the results of this paper to several network flow, knapsack, scheduling, and production planning problems. Many problems discussed in that paper have been previously considered with only a single criterion; previously known results for those problems are extended to the multi-criteria versions. The paper also treats the general integer version of some problems for which only the 0-1 case has been considered until now. In addition, fast approximation schemes are demonstrated for several problems that have not been discussed elsewhere.

1.1 Historical Perspective

Some papers that laid the foundations for this work are highlighted in this section. Detailed explanations of many of the topics considered here can be found in [GJ79].

Some early examples of approximation algorithms for $\mathcal{NP}$-hard problems are reviewed in [Joh74]. The concept of approximation scheme seems to have been introduced in [Sah75, IK75, Bab75]. These papers discuss simple knapsack problems; the first one presents PTAS's, and the others present FPTAS's, or FAS's, for these problems. These papers inspired a lot of subsequent research. The running times of FAS's for these knapsack problems were improved[GL79a, Law79]. FAS's were also found for other problems: for related knapsack problems[CHW76, Bab78, GL79b, Gen81], for multiprocessor scheduling problems[HS76, Sah76, GL78, GL81], and for some problems defined on trees[Sch83].

PTAS's are known for many problems for which FAS's are not known. PTAS's for several scheduling problems with precedence constraints appear in [IK78]. Approximation algorithms and PTAS's for a *continuous* knapsack problem are presented in [IHTI78, Iba80]. PTAS's for the multi-dimensional knapsack problem are described in [Fin75, Fin77]. It has been shown that this problem cannot have an FAS unless $\mathcal{P} = \mathcal{NP}$[GL79b, KS81, MC84]. Similar impossibility results for other problems are proved in [SG76, KK84].

Other existential questions about FAS's have also been investigated. The notion of *strong* $\mathcal{NP}$-hardness was introduced and it was shown that no strongly $\mathcal{NP}$-hard problem can have an FAS unless $\mathcal{P} = \mathcal{NP}$[GJ78]. The same work also proved that the existence of a pseudo-polynomial time optimization algorithm for a problem is a necessary condition for the existence of an FAS for the problem.

The existence of a pseudo-polynomial time algorithm is, however, not sufficient for the existence of an

FAS; understanding precise sufficiency conditions is one of the motivations for the present work. General techniques for finding an FAS are described in [Sah77], which also describes "dominance relations" between partial solutions that are sufficient to guarantee the existence of an FAS. These conditions are both necessary and sufficient for the existence of an FAS for a particular single-criterion maximization problem over an independence system[Kor79, KS81].

Many commonly treated optimization problems have a single criterion that is separable and linear. The existence of an algorithm from a particular subclass of pseudo-polynomial time algorithms is sufficient to guarantee the existence of an FAS for such a problem[PS82]. The present work generalizes this condition for the single criterion case and extends it to cover multi-criteria problems as well.

Multi-criteria optimization problems have received a lot of attention in the literature; surveys and explanations of results appear in [RV81, Ros85, Ste86]. Multi-criteria linear programs have been investigated in detail[Har85a]. Discrete multi-criteria problems have, however, long been recognized as being particularly difficult[Ser87]. Because of their importance, though, many solution approaches have been devised. The state of the art at various times is described in [SZ77, Zio78, Zio79, HM79, FG80, Ser85a].

One difficulty with multi-criteria problems is that, unlike in many single-criterion problems, efficient points of multi-criteria problems need not be at extreme points of the convex hull of the feasible set[Ser87]. Some approaches to dealing with this are described in [Sha76, Bit77, Bit79]. These problems are also difficult because even seemingly easy problems can have exponentially many solutions[EP88, Ruh88].

Dynamic programming is often used to solve discrete multi-criteria optimization problems[Whi69, Den82]. Techniques for extending traditional dynamic programming to multi-criteria problems are discussed in [BS65, DdK80, Hen83, Hen85a, Hen85b]. Unfortunately, the "curse of dimensionality" conspires to make a naive use of such algorithms impractical for large multi-criteria problems. Examples that explain this impracticality appear in [Whi80, Whi82a, Har85b]. Some successful applications are described in [VK81, VK82]. Conditions under which dynamic programming is useful are presented in [YS81, Har83].

Many related problems and approaches have been explored. Alternative definitions of efficiency are explored in [Geo68, Kal86]. A duality-based approach is described in [Ser84]; extensions to convex programs are discussed in [Ser85b]. An approach that converts the objective function into a scalar is presented in [Jah85]. The possibility of searching in the criteria space, rather than in the decision space, is examined in [AN79].

Special cases with particularly tractable objective functions are discussed in [Geo67, DW83]. Much work has been done on some problems with particularly simple structure. Shortest path problems have been notable in this regard[HZ80, Mar84a, Mar84b, Hen86, Whi82b], and a variety of other network optimization problems has also been treated[San86, SP87, Sni88, SOVM88]. In addition, knapsack problems[LM79] and scheduling problems[MP89] have been considered.

The existence of approximation schemes for some multi-criteria problems has been investigated. Early work along these lines was done on the shortest path problem[War83, War87]. Bicriteria minimum cost flow problems are the topic of [FBR89, RF89], and several path problems in directed graphs are covered in

[Han80]. General quasi-concave functions on convex regions are discussed in [KI87] and concave quadratic programming is discussed in [Vav90]. Fractional packing and covering problems have been investigated in [PST91].

General conditions for the existence of FAS's were presented in [Orl82], which forms the basis for this work. Somewhat similar techniques are used to find optimal solutions for single-criterion problems in [CGM89]. The present work generalizes those results to precisely identify necessary and sufficient conditions for the existence of an FAS for combinatorial optimization problems in both the single criterion and multi-criteria cases.

2 Motivation

This section discusses the difficulties that motivated this work. Some easy single-criterion problems become hard when more criteria are added; the example of the Shortest Path problem is given in Section 2.1. The solution to a multi-criteria problem can contain exponentially many points, as demonstrated in Section 2.2 using the Knapsack problem. Section 2.3 shows that the existence of a pseudo-polynomial time algorithm does not guarantee the existence of a fast approximation scheme.

2.1 Adding Criteria to Easy Single-Criterion Problems

Some single criterion problems that are easy to solve are difficult when multiple criteria are considered simultaneously. An example is the Shortest Path problem, which is polynomially solvable with a single criterion[Tar83, AMOT88, AMO93]. One reason the single criterion version of this problem is easy to solve is that the constraint matrix of a common integer programming formulation of the problem is totally unimodular (TUM)[Law76, PS82], so solving the linear programming relaxation of this formulation yields an optimal solution to the original problem.

A two-criteria version might have both a distance and a cost for each arc, with the goal of finding a path that is both short and inexpensive. Typically, though, no path is both the shortest and least expensive among all paths. The value associated with a feasible path, that is, the (distance, cost) pair it achieves, is *non-dominated* if every shorter feasible path is more expensive, and every less expensive feasible path is longer; that is, if no other feasible path is at least as good in both criteria and strictly better in at least one. Any path value that is not non-dominated must be *dominated* by the value of a path that is both shorter and less expensive.

The set of non-dominated values constitutes a *tradeoff curve* or *efficient frontier*, and a path whose value is non-dominated is called *efficient* or *Pareto optimal.* A solution to an instance of a multi-criteria shortest path problem is not just a single path, but rather a collection of efficient paths whose values define the efficient frontier.

The two-criteria Shortest Path problem is $\mathcal{NP}$-hard[GJ79, HZ80, Han80]. This is, at first, surprising, because total unimodularity depends only on the constraints, not on the objective function; adding an objective function would seem to not make a difference. The property that integrality constraints can be relaxed without penalty for problems defined by a TUM matrix does not hold when multiple criteria are considered simultaneously. The multiple criteria case causes at least three difficulties, even in the presence of total unimodularity.

- A point that is efficient among a set of integer-valued points need not be efficient when the intervening rational-valued points are considered as well, but this expansion of the feasible region is exactly what happens when the integrality constraints are relaxed. Furthermore, when considering rational-valued points, efficient points need not be at vertices of the feasible region. Example 1 demonstrates that the solution to the linear programming relaxation of a multi-criteria problem need not solve the problem.
- Example 2 shows that locally efficient points need not be globally efficient, even when considering only integer-value points.
- The number of efficient solutions can be exponential in the size of the problem, as illustrated in Section 2.2.

Example 1 (Non-extreme efficient points) One difficulty with multi-criteria problems is demonstrated using a simple example from [Ser87]; a similar example appears in [Sha76]. The problem is

$$\begin{array}{lrcr} \min & -1x & + & 3y \\ \min & 1x & - & 4y \end{array}$$

over the feasible region $[0,1]^2$, as shown in Figure 1.

The origin is efficient when considering only integer-valued points. If the integrality constraints were relaxed, however, $(0,0)$ would no longer be efficient; moving from any point along a ray with slope in the range $[\frac{1}{4}, \frac{1}{3}]$ improves (decreases) at least one objective function without increasing the other. All points in the hashed cone of Figure 1 dominate the origin, and the point $(1, \frac{7}{24})$ is an efficient point in that cone. In addition, all the points on the top and right sides of the square are efficient. ∎

Example 2 (Locally efficient points need not be globally efficient) A locally efficient point might not be globally efficient; an example is given in [Ser87]. Consider the objective functions

$$\begin{array}{lrcr} \min & -2x & + & 1y \\ \min & 1x & - & 2y \end{array}$$

over the feasible region $[0,1]^2$. Although the origin is locally efficient when considering only integer-valued points, it is not globally efficient, since it is dominated by $(1,1)$. ∎

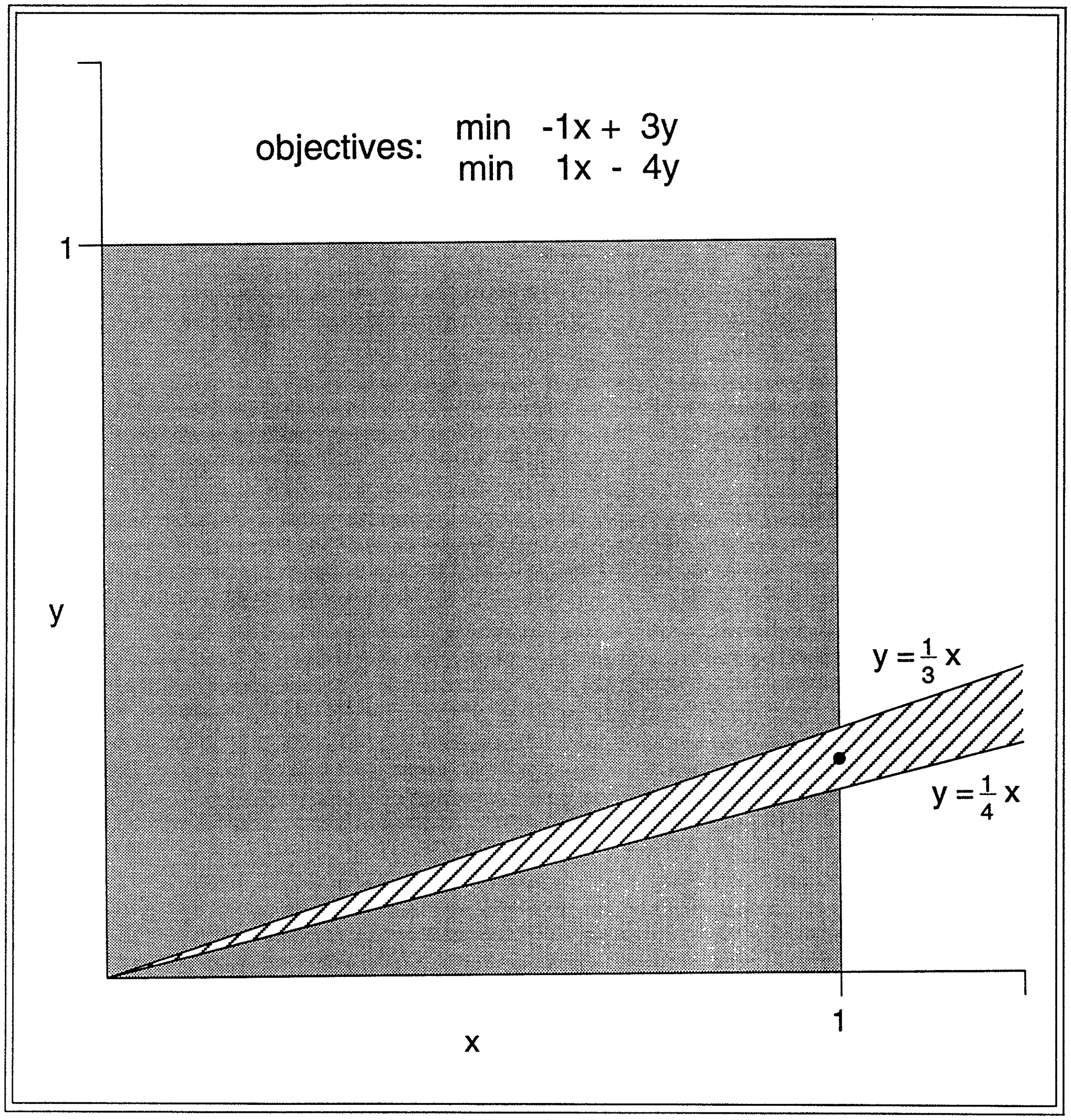

Figure 1: Non-extreme efficient points in the presence of total unimodularity.

The multi-criteria Shortest Path problem appears in many contexts; in particular, as pointed out in [Hen85a], it is a natural topic for investigations in multi-criteria dynamic programming. An early yet extensive consideration appears in [Han80], which discusses six kinds of objective functions: minsum, maxsum, minmax, maxmin, minproduct, and maxproduct. Various combinations of pairs are considered as objectives of bicriteria path problems. Many efficient optimization algorithms, approximation schemes, and proofs of difficulty are presented. Another kind of objective function is minratio, as discussed in [Meg79]. Most later works consider only problems in which all the criteria are of the minsum form; one exception is [Mar84b], in which one criterion is minmax and the second is either minmax, minsum, or minratio.

Many solution methods have been considered. A Lagrangian relaxation method for solving the bicriteria problem appears in [HZ80]; the second objective function is treated implicitly by turning it into a constraint. In [Mar84a], an algorithm reminiscent of the simplex method is used to move from one efficient solution to the next on the convex hull of the polyhedron of feasible solutions. A dynamic programming approach is explored in [Har85b]. Two criteria are combined using a single utility function in [Hen86], and algorithms are described for the quasi-concave and quasi-convex cases.

Except for [Han80], these works only discuss ways to find optimal solutions. An FAS is presented in [War83, War87]; the results in Section 4 generalize the results from these two papers.

2.2 Optimal Solutions can be Large

The solution to a multi-criteria problem can contain exponentially many points. This section presents two examples in which this is true. The first example shows that the number of solutions can grow exponentially with the number of criteria. The second example shows that the number of solutions can grow exponentially with the problem size even if the number of criteria is fixed. The examples use variants of the Max Binary Knapsack problem. This problem is $\mathcal{NP}$-hard[GJ79], but it has an FAS[IK75, Bab75].

Max Binary Knapsack

Instance: (n, p, V, v). The number of items n; the profits p_j, $j = 1, \ldots, n$; the knapsack volume V; and the item volumes v_j, $j = 1, \ldots, n$; are non-negative integers such that $v_j \leq V$, $j = 1, \ldots, n$.

Question: Find x to solve

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} p_j x_j & & \\ \text{s.t.} \quad & \sum_{j=1}^{n} v_j x_j & \leq & \; V \\ & \qquad x_j & \in & \; \{0,1\}, \; j = 1, \ldots, n \; . \end{aligned}$$

Example 3 (Solution size exponential in number of criteria) The size of the efficient frontier for a multi-criteria problem can grow exponentially with the number of criteria, even if the objective function coefficients are small. This phenomenon is demonstrated here using a knapsack problem; a similar example using a shortest path problem is alluded to in [War87].

$$\begin{array}{llllll}
\textbf{max}\text{ obj. 1:} & x_{1,1}+\cdots+x_{1,n} & & & & \\
\textbf{max}\text{ obj. 2:} & & x_{2,1}+\cdots+x_{2,n} & & & \\
\vdots & & & \ddots & & \\
\textbf{max}\text{ obj. } r\text{:} & & & & x_{r,1}+\cdots+x_{r,n} & \\
 & & & & & \\
\text{subject to:} & x_{1,1}+\cdots+x_{1,n} \;+ & x_{2,1}+\cdots+x_{2,n} & +\cdots+ & x_{r,1}+\cdots+x_{r,n} & \leq n \\
 & \multicolumn{5}{l}{x_{i,k} \in \{0,1\}\,,\ i=1,\ldots,r,\ k=1,\ldots,n}
\end{array}$$

Figure 2: Instance (r, n) of Example 3.

Consider an r-criteria Max Binary Knapsack problem, with each criterion to be maximized. Define an infinite family of instances indexed by r and n. Instance (r, n) contains rn variables and all the coefficients are either 0 or 1, yet as will be shown below, the instance has $\binom{n+r-1}{n} \geq n^{r-1}$ efficient points. This is exponential in the number of criteria r.

To be specific, instance (r, n) has variables $x_{i,k}$, $j = 1, \ldots, r$ and $k = 1, \ldots, n$, with $x_{i,k} \in \{0,1\}$ for all i and k. The only constraint is the knapsack constraint

$$\sum_{i=1}^{r}\sum_{k=1}^{n} x_{i,k} \leq n \ .$$

Partition the variables into r groups of n variables each; the i^{th} group contains the variables $x_{i,k}$, $k = 1, \ldots, n$. The i^{th} objective function, $i = 1, \ldots, r$, is the sum of the variables in the i^{th} set, that is, $f_i(x) = \sum_{k=1}^{n} x_{i,k}$. The value of $f_i(x)$ is therefore the number of variables $x_{i,k}$ that have been set to 1 in the solution x. The formulation is illustrated in Figure 2.

Any efficient solution must satisfy the knapsack constraint at equality; given a feasible point in which fewer than n variables have been set to 1, and in which $x_{\bar{i},\bar{k}} = 0$, setting $x_{\bar{i},\bar{k}} = 1$ will strictly improve the value of the i^{th} objective function without changing the value of any other objective function. So for each efficient point exactly n variables have value 1, and so for each efficient point $\sum_{i=1}^{r} f_i(x) = n$. Furthermore, any way of partitioning the n knapsack units among the r criteria can be achieved by some feasible point. Any solution must therefore contain a point corresponding to each possible partition of the n units. The number of partitions is $\binom{n+r-1}{n} \geq n^{r-1}$[Fel68, Chapter II.5], so the number of points on the efficient frontier is exponential in the number of criteria. ■

Problems in which the number of criteria grow linearly with the problem size are not solvable by FAS's, with the exception of members of some trivial problem classes. The present work therefore considers only problems in which the number of criteria is fixed. Even so, the number of efficient points can be exponentially large in the number of variables, as in the following example.

Example 4 (Solution size exponential in number of variables) The Arborescent Knapsack problem is a variant of the Max Binary Knapsack problem in which some items are stored in sub-knapsacks or

$$
\begin{array}{llcccccccl}
\textbf{max}\text{ obj. 1:} & & 2x_{1,2} & + & & 4x_{2,2} & + & \cdots & + & 2^n x_{n,2} \\
\textbf{max}\text{ obj. 2:} & 2x_{1,1} & & + & 4x_{2,1} & & + & \cdots & + & 2^n x_{n,1} \\
\\
\text{subject to:} & x_{1,1} + x_{1,2} & & & & & & & & \leq 1 \\
 & & & & x_{2,1} + x_{2,2} & & & & & \leq 1 \\
 & & & & & & \ddots & & & \vdots \\
 & & & & & & & & x_{n,1} + x_{n,2} & \leq 1 \\
 & x_{1,1} + x_{1,2} & & + & x_{2,1} + x_{2,2} & & + & \cdots & + \; x_{n,1} + x_{n,2} & \leq n \\
 & x_{j,1}, x_{j,2} \in \{0,1\}, \; j = 1, \ldots, n
\end{array}
$$

Figure 3: Instance n of Example 4.

stuffsacks; it is discussed in more detail in [SO94]. Two stuffsacks either have no items in common, or else one is wholly contained in the other. A knapsack constraint is specified for each stuffsack as well as for the entire knapsack.

Define the following infinite family of two-criteria instances, indexed by n; recall that the number of criteria is fixed. The n^{th} instance has $2n$ items, represented by $x_{j,1}$ and $x_{j,2}$, $j = 1, \ldots, n$. The vector of objective value coefficients for $x_{j,1}$ is $\binom{0}{2^j}$ and for $x_{j,2}$ is $\binom{2^j}{0}$, and both objective functions are to be maximized.

Items $x_{j,1}$ and $x_{j,2}$ may be put in a stuffsack S_j, and at most one of each pair can be chosen. This means that at most n variables are to be chosen overall. The formulation is shown in Figure 3.

All constraints will be satisfied at equality for an efficient point; if stuffsack S_j is empty in some solution, then putting either $x_{j,1}$ or $x_{j,2}$ into the stuffsack would yield a feasible point that strictly improves one objective function without affecting the other.

An instance of this problem has 2^n efficient points; that is, each possible way of choosing one variable from each pair $x_{j,1}$ and $x_{j,2}$ yields a different objective value, and no value dominates another. To see this, consider any two different feasible points y and z. The values corresponding to y and z must differ in the first objective, but the sum of the two objectives is $2^{n+1} - 2$ at each point. Therefore the value of y does not dominate the value of z, nor does the value of z dominate the value of y. ■

2.3 A Pseudo-Polynomial Time Algorithm is Not Sufficient for an FAS

An algorithm runs in *pseudo-polynomial time* if it solves any instance in time polynomial in the size of the instance and in the value of the largest integer in the instance description. It is well known that the existence of a pseudo-polynomial time algorithm is necessary for the existence of an FAS for the problem[GJ78]. If $\mathcal{P} \neq \mathcal{NP}$, however, then this condition is not sufficient, but this is not as widely recognized. [PS82] identifies a subclass of the pseudo-polynomial time algorithms; the existence of an algorithm from this subclass for a problem is sufficient to guarantee the existence of an FAS for the problem. This section explains why the restriction to such a subclass is necessary. Section 4 generalizes the condition of [PS82] to cover more general objective functions and multi-criteria problems.

An indication that the existence of a pseudo-polynomial algorithm is not sufficient to guarantee the existence of an FAS if $\mathcal{P} \neq \mathcal{NP}$ is contained in [Law79, Section 16]. The discussion there concerns the following optimization problem:

Max 2-Dimensional Binary Knapsack

Instance: (n, p, V, v, W, w). The number of items, n; the profits p_j, $j = 1, \ldots, n$; the knapsack volume V; the item volumes v_j, $j = 1, \ldots, n$; the knapsack weight capacity W; and the item weights w_j, $j = 1, \ldots, n$; are non-negative integers such that $v_j \leq V$ and $w_j \leq W$, $j = 1, \ldots, n$.

Question: Find x to solve

$$\begin{array}{rlcl} \max & \sum_{j=1}^{n} p_j x_j & & \\ \text{s.t.} & \sum_{j=1}^{n} v_j x_j & \leq & V \\ & \sum_{j=1}^{n} w_j x_j & \leq & W \\ & x_j & \in & \{0,1\},\ j = 1, \ldots, n\ . \end{array}$$

An instance of this problem can be solved by a simple pseudo-polynomial time algorithm that runs in time $O\left(n^2 p_{max} V\right)$, where p_{max} is the largest p_j value, $j = 1, \ldots, n$. More sophisticated dynamic programming approaches to this problem are presented in [ME74, MM76, MM78]. Nonetheless, this problem cannot have an FAS unless $\mathcal{P} = \mathcal{NP}$[GL79b, KS81, MC84].

As another example, consider the following optimization problem:

Max Equality-Constrained Binary Knapsack

Instance: (n, p, V, v). The number of items n; the profits p_j, $j = 1, \ldots, n$; the knapsack volume V; and the item volumes v_j, $j = 1, \ldots, n$; are non-negative integers such that $v_j \leq V$, $j = 1, \ldots, n$.

Question: Find x to solve

$$\begin{array}{rlcl} \max & \sum_{j=1}^{n} p_j x_j & & \\ \text{s.t.} & \sum_{j=1}^{n} v_j x_j & = & V \\ & x_j & \in & \{0,1\},\ j = 1, \ldots, n\ . \end{array}$$

An instance of this problem can be solved by a pseudo-polynomial time algorithm that runs in time $O\left(n p_{max} V\right)$. As with the previous problem, however, this problem cannot have an FAS unless $\mathcal{P} = \mathcal{NP}$, since finding a single feasible solution is the $\mathcal{NP}$-hard Subset Sum problem[GJ79].

These examples show that the well-known necessary condition for the existence of an FAS for a problem, the existence of a pseudo-polynomial time algorithm for the problem, must be restricted in order to guarantee the existence of an FAS for the problem. Section 4 identifies the appropriate restriction precisely. First, though, simplifying notation is presented.

3 Terminology

This section introduces the terminology and notation used in the rest of the paper. Section 3.1 shows how to specify instances and problems. Section 3.2 defines two characteristics of an instance that are used to measure the computational complexity of solution algorithms. Sections 3.3 and 3.4 describe exact and approximate solutions, respectively, and explain characteristics of algorithms that find such solutions.

The following conventions are observed, except where specified. Logarithms are to the base 2. Vector norms are infinity norms, that is, $||x|| = \max\{|x_j|\}$. The vector e_k is a vector of length k in which each component is one. Some terminology regarding solutions is adapted from [War87].

3.1 Representing Instances and Problems

This section describes the representation of optimization and feasibility problems. Informally, an optimization problem consists of instances, each of which is solved by a set of points whose objective values define an efficient frontier. A *feasibility* problem consists of instances, each of which is solved by a single point whose objective value achieves a specified target value.

An optimization instance is represented by a 3-tuple that specifies a feasible set, an objective function, and a direction for each criterion, i.e., minimization or maximization. A feasibility instance is represented by a 4-tuple with the same three entries as well as a target value. The definitions of these concepts are followed by an illustrative example.

Feasible Sets. An n-dimensional or n-variable *feasible set* is a bounded set $S \subseteq \mathcal{Z}^{n+}$, where $\mathcal{Z}^+$ is the set of non-negative integers. A feasible set is often described by an integer program, but this representation is not required for the results presented here. Because S is bounded, each variable can be considered as having a simple upper bound constraint. In other words, the representation of a feasible set S contains explicit or implicit constraints of the form $x \leq u(S)$, where $u(S) \in \mathcal{Z}^{n+}$. Each element of $u(S)$ is as small as possible, that is, the upper bound constraints are tight. The largest element of $u(S)$ is $u_{max}(S) = ||u(S)||$.

A *family of feasible sets*, represented by Ψ, may contain feasible sets of different dimensions. The *domain* of Ψ is $\{0, \ldots, U\}$, where U is the largest value of $u_{max}(S)$ over all feasible sets $S \in \Psi$. A family is *binary* if its domain is $\{0, 1\}$.

Objective Functions. An r-criteria *objective function* f maps a feasible set S into $\mathcal{Z}^{r+}$. It is made up of r component functions $f_i : S \rightarrow \mathcal{Z}^+$. An objective function is assumed to satisfy the following two conditions. These assumptions are needed for the proofs of the main theorems, in Section 4, and are typically satisfied by objective functions encountered in practice.

The first condition just says that an objective function can be evaluated within a reasonable length of time. It holds for polynomial functions as well as for other useful classes of functions that are discussed in [SO94]. If computing $f(\cdot)$ takes longer than this length of time, then f cannot generally be evaluated as part of an effective solution procedure anyhow. An alternative, but less realistic, approach would be to let f be an oracle function and allow its evaluation in one step[GJ79, HU79].

Assumption 1 *An r-criteria objective function f can be evaluated at a point $x \in S \subseteq \mathcal{Z}^{n+}$ in time polynomial in the sizes of x and $f(x)$, that is, in time $O\left([n \cdot r \cdot \log(\|x\|) \cdot \log(\|f(x)\|)]^k\right)$, for some $k \in \mathcal{Z}^+$.*

The second assumption says that determining if the value of an objective function is within some bound can be achieved within a reasonable length of time. This is a technical condition that is needed in the development of the sufficient conditions for the existence of an FAS. As with the previous assumption, it is true of most commonly-used objective functions. Furthermore, the method for answering the question need be no more complicated than an algorithm to compute the function $f_i(\cdot)$. Once the particular polynomial time bound for f_i is determined, the computation of $f_i(x)$ can be timed; if the time bound is reached, then the conclusion that the bound is exceeded may be drawn.

Assumption 2 *For any objective function component $f_i : S \to \mathcal{Z}^+$, $M_i \in \mathcal{Z}^+$, and $x \in S$, the truth of the statement "$f_i(x) \leq M_i$" can be determined in time polynomial in the sizes of x and M_i, that is, in time $O\left([n \cdot \log(\|x\|) \cdot \log(M_i)]^k\right)$, for some $k \in \mathcal{Z}^+$.*

A *family of r-criteria objective functions* is denoted by $\mathcal{F}$. Although a family Ψ of feasible sets may contain feasible sets of different dimensions, the functions in $\mathcal{F}$ *all* have r criteria. If r were allowed to vary, many problems considered here would be intractable, as demonstrated in Example 3 of Section 2.2. Because of the exponential dependence of the running time on the number of criteria, the existence of an FAS need not imply the existence of an approximation scheme that works well in practice.

Directions of Optimization. Suppose that $\mathcal{F}$ is a family of r-criteria functions. The direction in which to optimize each criterion is specified by an r-dimensional *direction vector* with entries from the set $\{" \leq ", " \geq ", " = "\}$. Intuitively, an entry of "$\leq$" for an optimization problem means that the corresponding component of the objective function is to be *minimized*; for a feasibility problem, it means that the value of the corresponding component must be *no more than* the target value. The use of a direction vector is explained more precisely when solutions are defined in Sections 3.3 and 3.4.

A direction vector is generally represented by ω. The symbols $\omega^{r,<}$, $\omega^{r,>}$, and $\omega^{r,=}$ represent r-dimensional direction vectors with a single kind of entry. The set Ω contains all direction vectors and Ω^r contains all direction vectors of length r. The sets $\Omega^<$, $\Omega^>$, and $\Omega^=$ contain $\omega^{r,<}$, $\omega^{r,>}$, and $\omega^{r,=}$, respectively, for all $r \in \mathcal{Z}^+$.

Instances and Problems. An *instance of an optimization problem* is represented by a 3-tuple $I = (S, f, \omega)$, where for positive integers n and r, S is an n-dimensional feasible region, f is an r-criteria objective function, and ω is an r-dimensional direction vector. An *optimization problem* $\Pi = (\Psi, \mathcal{F}, \omega, \mathrm{Opt})$ is a collection of instances $(S, f, \overline{\omega})$ with $S \in \Psi$, $f \in \mathcal{F}$, and $\overline{\omega} = \omega$.

An *instance of a feasibility problem* is represented by a 4-tuple $I = (S, f, \omega, M)$. The first three components are the same as for an optimization instance. The fourth component is $M \in \mathcal{Z}^{r+}$, the *vector of target values.* The precise use of M is explained in Section 3.3. A *feasibility problem* $\Pi = (\Psi, \mathcal{F}, \omega, \text{Feas})$ is a collection of instances $(S, f, \overline{\omega}, M)$ with $S \in \Psi$, $f \in \mathcal{F}$, $\overline{\omega} = \omega$, and $M \in \mathcal{Z}^{r+}$.

If Π is a problem of either kind, then $I \in \Pi$ means that I is an instance of Π.

Example 5 (Instance and problem definitions) The definitions from this section are illustrated using a knapsack problem in which each item contains both a desirable and an undesirable feature. This problem can be formulated as follows:

<u>Max-Min Binary Knapsack</u>

Instance: (n, p, q, V, v). The number of items n; the profits p_j, $j = 1, \ldots, n$; the costs q_j, $j = 1, \ldots, n$; the knapsack volume V; and the item volumes v_j, $j = 1, \ldots, n$; are non-negative integers such that $v_j \leq V$, $j = 1, \ldots, n$.

Question: Find x to solve

$$\begin{array}{lrcl} \max & \sum_{j=1}^{n} p_j x_j & & \\ \min & \sum_{j=1}^{n} q_j x_j & & \\ \text{s.t.} & \sum_{j=1}^{n} v_j x_j & \leq & V \\ & x_j & \in & \{0,1\},\ j = 1, \ldots, n \end{array}$$

The feasible set for an instance of this problem can be specified as

$$S(n, V, v) = \left\{ x \in \{0,1\}^n \, : \, \sum_{j=1}^{n} v_j x_j \leq V \right\} ,$$

the upper bounds are

$$u(S) = e_n \ ,$$

the objective function has the form

$$f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \end{pmatrix} = \begin{pmatrix} f_1(x;\ n, p) \\ f_2(x;\ n, q) \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{n} p_j x_j \\ \sum_{j=1}^{n} q_j x_j \end{pmatrix} ,$$

and the direction vector is

$$\omega = \begin{pmatrix} \omega^{1,>} \\ \omega^{1,<} \end{pmatrix} = \begin{pmatrix} \text{“} \geq \text{”} \\ \text{“} \leq \text{”} \end{pmatrix} .$$

The optimization form of the problem can be written as $\Pi = (\Psi, \mathcal{F}, \omega, \text{Opt})$, where

$$\Psi = \left\{ S(n, V, v) \, : \, n, V \in \mathcal{Z}^{+},\ v \in \mathcal{Z}^{n+},\ \text{and } v_j \leq V,\ j = 1, \ldots, n \right\}$$

and

$$\mathcal{F} = \left\{ \begin{pmatrix} f_1(x;\ n,p) \\ f_2(x;\ n,q) \end{pmatrix} : n \in \mathcal{Z}^+,\ p \in \mathcal{Z}^{n+},\ q \in \mathcal{Z}^{n+} \right\} .$$

Note that Ψ is binary.

The related feasibility problem is the set of instances (S, f, ω, M) where S, f, and ω are as above, and M is any vector in $\mathcal{Z}^{2+}$. ■

3.2 The Largest Value and the Length of an Instance

The *largest value* of an optimization instance $I = (S, f, \omega)$ is the largest component of the objective function on a particular box-constrained superset of the feasible region:

$$M_V(I) = \max\{ \|f(x)\| \ : \ 0 \leq x \leq u(S) \} .$$

In a feasibility instance $I = (S, f, \omega, M)$, values of $f(x)$ with $\|f(x)\| > M$ need not be considered, so the largest value of I is $M_V(I) = \|M\|$. The subscript "v" is a reminder that this is the largest value of the instance. It is not the largest number of the instance that is used in discussions of pseudo-polynomial time algorithms.

The *length* of an n-variable instance I of either kind, written $L(I)$, is the number of bits needed to represent both S and $M_V(I)$.

In order to avoid the intricacies required to explicitly address the representation of general functions, the length of an instance is defined independently of the representation of the objective function. This does not result in the loss of much generality, because most interesting objective functions can be represented in space that is polynomial in the length of the instance as defined here. The objective functions could be added by recasting the discussion in terms of Turing machines[HU79]. The added complexity of the presentation, however, would not add insight to the results.

In addition, the length of an instance is independent of the number of criteria in its objective function. Accounting for this number would increase the length by at most a constant factor since, as explained in Sections 2.2 and 3.1, all the instances of a problem have the same number of criteria.

Although $M_V(I)$ may be difficult to determine exactly, it is assumed that it can be quickly approximated to within a factor that is polynomial in the length of the instance.

Assumption 3 *For any instance I, a close upper bound $\overline{M}_v(I)$ of $M_v(I)$ can be found in time polynomial in the length $L(I)$ of the instance. Here "close" means within a factor that is polynomial in $L(I)$. That is, for some $k_1, k_2 \in \mathcal{Z}^+$, time $O\left([L(I)]^{k_1}\right)$ is sufficient to find a bound $\overline{M}_v(I)$ satisfying*

$$M_v(I) \leq \overline{M}_v(I) \in O\left(M_v(I)\,[L(I)]^{k_2}\right) . \tag{1}$$

In most interesting discrete optimization problems, a bound of this sort can be found easily. Creative approaches to finding such bounds are described in [IK75, Law79].

Example 6 (Largest value and length of an instance) The Max-Min Binary Knapsack problem was discussed in Example 5. An instance $I = (S(n, V, v), f(x;\ n, p), \omega)$ has largest value

$$\begin{aligned} M_V(I) &= \max\left\{\sum_{j=1}^{n} p_j,\ \sum_{j=1}^{n} q_j\right\} \\ &= n \max\{p_{max},\ q_{max}\} \end{aligned}$$

and length

$$L(I) \in O(n \log(V) + n\{\log(p_{max}) + \log(q_{max})\}) \ .$$

The value $\overline{M}_V(I) = M_V(I)$ can be found in time $O(n) \subseteq O(L(I))$ by scanning the components of p and q. ■

3.3 Exact Solutions and Algorithms

Given a direction vector $\omega \in \Omega^r$ and two vectors $u, v \in \mathcal{Z}^{r+}$, u *dominates* v with respect to (w.r.t.) ω if u is at least as good a value as v in each component in the direction specified by ω. So if ω_i is " $\leq$ ", then $u_i \leq v_i$ if u dominates v, and similarly for other values of ω_i. This relation is expressed using the predicate $D_\omega(u, v)$, which means that, for $i = 1, \ldots, r$,

$$\begin{cases} \omega_i = \text{“} \leq \text{”} & \Rightarrow \quad u_i \leq v_i \quad , \\ \omega_i = \text{“} \geq \text{”} & \Rightarrow \quad u_i \geq v_i \quad , \\ \omega_i = \text{“} = \text{”} & \Rightarrow \quad u_i = v_i \quad . \end{cases}$$

Keep in mind that u and v in this definition are in criterion space, that is, they are in the space of objective function values.

A solution to an optimization instance $I = (S, f, \omega)$ is a set of feasible points whose values constitute the efficient frontier. This is the second of the approaches to vector optimization listed in [Ser87]. More precisely, an *efficient set* for I is a set $Y \subseteq S$ such that for each $x \in S$, there is some $y \in Y$, depending on x, such that y is at least as efficient as x in the sense specified by ω; that is, $D_\omega(f(y), f(x))$. A *solution* to I is a minimum cardinality efficient set for I. Here x and y are in decision space, and $f(x)$ and $f(y)$ are in criterion space.

A solution to a feasibility instance $I = (S, f, \omega, M)$ is a point $x \in S$ such that $D_\omega(f(x), M)$, if such a point exists. In other words, it is any element of the set of feasible points whose values dominate M. Note that a feasibility instance is not a recognition instance. A solution to the former is an appropriate feasible point; a solution to the latter is just a statement that an appropriate feasible point exists[GJ79].

An algorithm for an optimization or a feasibility problem Π is *V-pseudo-polynomial*, or VPP, if for any instance $I \in \Pi$, the algorithm finds a solution for I in time polynomial in the length and largest value of the instance, that is, in time $O\left(\left[L(I) \cdot M_V(I)\right]^k\right)$, for some $k \in \mathcal{Z}^+$. The "V" in VPP stands for "value." If some VPP algorithm exists for Π, Π is said to "have a VPP algorithm." Any VPP algorithm is also pseudo-polynomial, but the converse is not true.

Example 7 (Exact solutions) Recall the Max-Min Binary Knapsack problem discussed in Examples 5 and 6. Consider the instance with $n = 8$, $V = 3$, and the following coefficients:

j	1	2	3	4	5	6	7	8
p_j	1	2	4	5	6	7	9	11
q_j	2	2	4	4	5	6	10	9
v_j	2	1	1	1	2	2	3	3

Figure 4 shows all the values of the objective function at feasible points. The point $\binom{6}{5}$ dominates the point $\binom{5}{6}$, since the first objective is a profit and the second is a cost.

The solid dots constitute the efficient frontier. An efficient set is a collection of feasible points whose values are shown by the solid dots. If $x_1 = (0,0,0,0,0,0,0,1)$ and $x_2 = (0,0,0,1,1,0,0,0)$, then $f(x_1) = f(x_2) = \binom{11}{9}$; either x_1 or x_2, but not both, will occur in a solution, since a solution is an efficient set of minimum cardinality.

Let $x_3 = (0,0,1,0,0,0,0,0)$ and $x_4 = (0,0,0,1,0,0,0,0)$. The point $\binom{4}{5}$ is dominated by both $f(x_3) = \binom{4}{4}$ and $f(x_4) = \binom{5}{4}$. So if $M = \binom{3}{4}$ in an instance of the feasibility form of this problem, x_3 and x_4 are each solutions, even though x_5 is not efficient. ■

3.4 Approximate Solutions and Algorithms

This section defines concepts for approximate solutions that are analogous to those for exact solutions in Section 3.3. Given a direction vector $\omega \in \Omega^r$, two vectors $u, v \in \mathcal{Z}^{r+}$, and $\varepsilon \geq 0$, u *ε-dominates* v w.r.t. ω if each component of u is no more than a factor of ε worse than the corresponding component of v in the direction specified by ω. So if ω_i is " $\leq$ ", then $u_i \leq (1+\varepsilon)v_i$ if u ε-dominates v. This relation is expressed using the predicate $D_\omega^{(\varepsilon)}(u,v)$, which means that, for $i = 1, \ldots, r$,

$$\begin{cases} \omega_i = \text{“}\leq\text{”} & \Rightarrow & & & u_i & \leq & (1+\varepsilon)v_i \quad , \\ \omega_i = \text{“}\geq\text{”} & \Rightarrow & (1-\varepsilon)v_i & \leq & u_i & & \quad , \\ \omega_i = \text{“}=\text{”} & \Rightarrow & (1-\varepsilon)v_i & \leq & u_i & \leq & (1+\varepsilon)v_i \quad . \end{cases}$$

When using this notation, recall from Section 3.1 that objective function values are non-negative. When $\varepsilon = 0$, then ε-domination reduces to domination; that is, $D_\omega^{(0)}(u,v)$ is the same as $D_\omega(u,v)$.

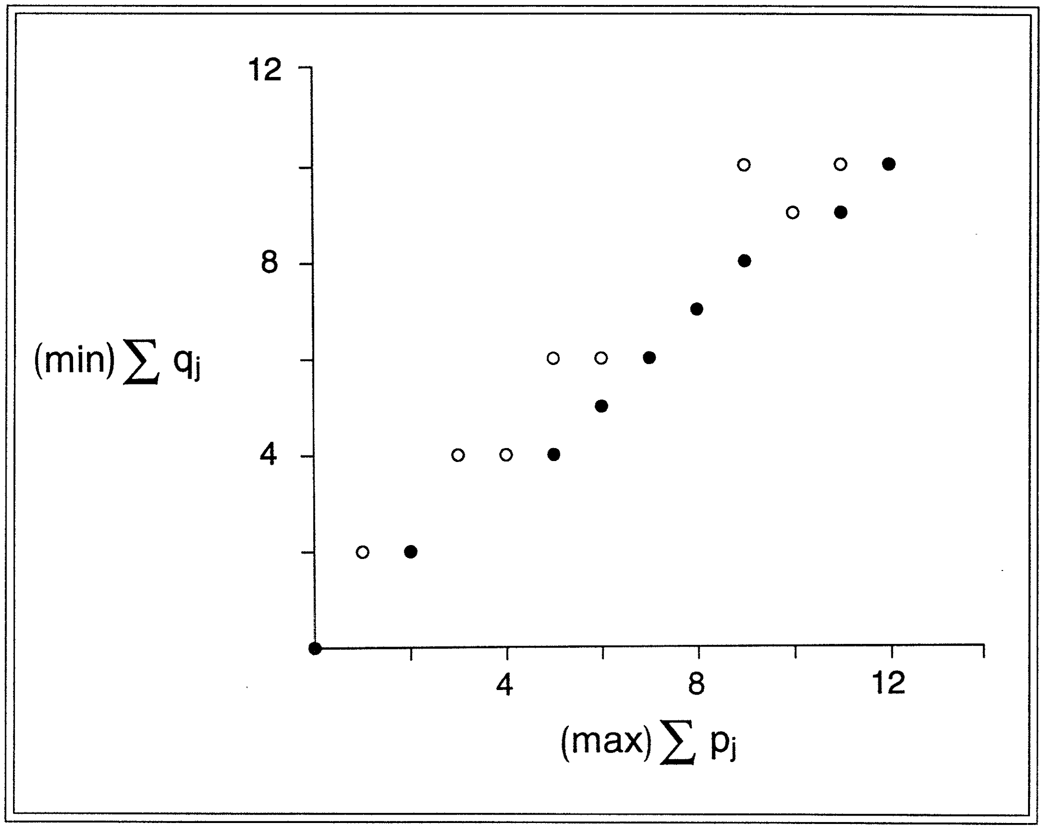

Figure 4: Feasible solution values for the instance in Example 7. The solid dots constitute the efficient frontier.

For an r-vector v and any $\varepsilon > 0$, call the set of points that are within a factor ε of v in each component an *ε-ball* around v. Figure 5 shows $\frac{1}{4}$-balls around the points on the efficient frontier of Figure 4. The balls are rectangles because the difference from v is measured separately for each dimension. The balls grow bigger as the values increase because ε is a measure of relative error from v, not absolute error.

An ε-efficient solution to an optimization instance is a set of feasible points with the property that each point of the efficient frontier is close to, or approximated by, the objective function value of at least one point in the set. In other words, an ε-ball drawn around any point on the efficient frontier contains the objective function value of at least one point of the solution set. More formally, an *ε-efficient set* for an instance $I = (S, f, \omega)$ is a set $Y \subseteq S$ such that for each $x \in S$, there is some $y \in Y$, depending on x, such that $D_{\omega}^{(\varepsilon)}(f(y), f(x))$. An *$\varepsilon$-efficient solution* to I is a minimal ε-efficient set for I.

Note that an ε-efficient solution is minimal with respect to the property of ε-efficiency. It need not have minimum cardinality, as will be explained in Example 8, but by definition it must not have a proper subset that is ε-efficient. If $\varepsilon = 0$, however, then any minimal exact solution will also have minimum cardinality; the solution will consist of one efficient point for each point on the efficient frontier.

The notion of ε-efficiency is defined only for optimization problems, not for feasibility problems. An algorithm for an optimization problem $\Pi = (\Psi, \mathcal{F}, \omega, \mathrm{Opt})$ is a *fast approximation scheme*, or FAS, if for any $\varepsilon > 0$ and any instance $I \in \Pi$, the algorithm finds an ε-efficient solution for I in time polynomial in the length of the instance and in $1/\varepsilon$, that is, in time $O\left([L(I)/\varepsilon]^k\right)$, for some $k \in \mathcal{Z}^+$. If some FAS exists for Π, Π is said to "have an FAS." When restricted to the single-criterion problems commonly considered in work on approximation algorithms, an FAS is a fully polynomial time approximation scheme (FPTAS)[GJ79, PS82].

In these definitions the ε-balls are centered on the points of the efficient frontier, so that each efficient point is sure to be approximated. One could instead think of centering the ε-balls on the approximate solutions or use alternative approaches similar to those described in [Whi86].

Although widely used in single-criterion optimization, this approximation measure is sometimes problematic, such as in a minimization problem whose optimal solution can have the value zero. This issue is discussed in detail in [CFN77, NWF78, Fis80, Zem81], and is not addressed further here.

Example 8 (Approximate solutions) Approximate solutions are demonstrated using the instance discussed in Example 7. If $\varepsilon = \frac{1}{9}$, then $\binom{11}{10}$ $\frac{1}{9}$-dominates both $\binom{11}{9}$ and $\binom{12}{10}$. To see the first of these, notice that $11 \geq (1-\varepsilon)11$ and $10 \leq (1+\varepsilon)9$. However, $\binom{6}{6}$ does not $\frac{1}{9}$-dominate either $\binom{6}{5}$ or $\binom{7}{6}$, even though these two points are the same absolute distance from $\binom{6}{6}$ as $\binom{11}{9}$ and $\binom{12}{10}$ are from $\binom{11}{10}$. Points in an approximate solution may or may not be on the efficient frontier; for example, consider the elements of A_2 discussed in the next paragraph.

An ε-efficient solution is an ε-efficient set of *minimal* cardinality, whereas an exact solution is an efficient set of *minimum* cardinality. To see that two minimal ε-efficient solutions need not have the same size, consider the set C of efficient points other than $\binom{0}{0}$ and $\binom{2}{2}$ in Figure 5. Each point of C is $\frac{1}{4}$-dominated by some point of $A_1 = \left\{\binom{6}{5}, \binom{11}{9}\right\}$, and also by some point of $A_2 = \left\{\binom{5}{4}, \binom{6}{6}, \binom{10}{9}\right\}$. Removing any point from either

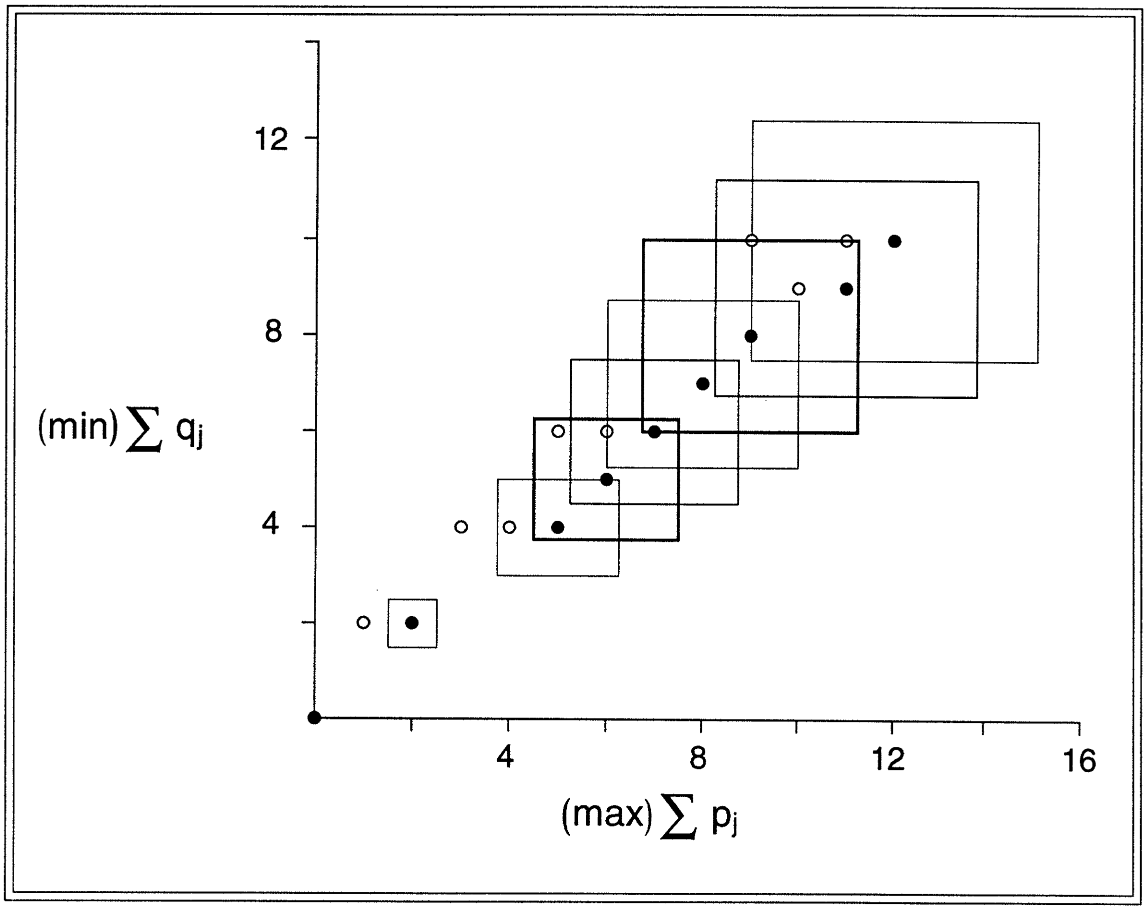

Figure 5: Feasible solution values and $\frac{1}{4}$-balls around points on the efficient frontier for the instance in Example 8.

A_1 or A_2, however, would destroy the $\frac{1}{4}$-dominance property of that set. Therefore both are minimal in this respect, even though they have different numbers of elements. ■

4 Existence of Fast Approximation Schemes

The primary theoretical results of this paper are derived in this section. A necessary condition for the existence of a fast approximation scheme for a problem is stated and proved in Section 4.1. This theorem is followed by an example that demonstrates that its converse is false, but Section 4.2 develops conditions under which the converse is true. A theorem about the sufficient condition for the existence of an FAS is formally stated and proved in Section 4.3. Finally, some aspects of the relationship of this work to previous work are discussed in Section 4.4.

4.1 Necessary Condition

A necessary condition for the existence of an FAS for a problem is the existence of a VPP algorithm for the problem. The proof consists of showing how an FAS for the problem can be used to create a VPP algorithm. This is done by setting the error tolerance for the FAS small enough so that for any efficient point x, some element y of the ε-efficient solution has an *absolute* error from x that is strictly smaller than unity in each criterion. Since the function values are integral, this means that x and y have identical function values; in other words, an exact solution will have been found.

Theorem 1 (Necessary condition for existence of an FAS) *Consider an optimization problem* $\Pi = (\Psi, \mathcal{F}, \omega, Opt)$. *If* Π *has an FAS, then* Π *has a VPP algorithm.*

Proof. Suppose that an FAS $\mathcal{H}$ for Π is specified. The following algorithm will be shown to be a VPP algorithm for solving Π. Let $I = (S, f, \omega)$ be an n-variable instance of Π.

1. Determine $\overline{M}_V(I)$ satisfying Equation 1 and set $\varepsilon \leftarrow \frac{1}{1+\overline{M}_V(I)}$.
2. Use $\mathcal{H}$ to solve I with accuracy ε.

The proof that this is a VPP algorithm for Π has two parts: first the algorithm is proved to run in VPP time, then the solution it finds is shown to be correct.

Running time: By Assumption 3, Step 1 takes time $O\left([L(I)]^{k_1}\right)$, for some $k_1 \in \mathcal{Z}^+$. Because $\mathcal{H}$ is an FAS for Π, Step 2 takes time $O\left([L(I)/\varepsilon]^{k_2}\right) \subseteq O\left([L(I) \cdot M_V(I)]^{k_3}\right)$, for some $k_2, k_3 \in \mathcal{Z}^+$. The algorithm therefore runs in VPP time.

Accuracy: Suppose that the algorithm returns a set $Y \subseteq S$; then for any $x \in S$, some $y \in Y$ satisfies $D_\omega^{(\varepsilon)}(f(y), f(x))$. It will be shown that $D_\omega(f(y), f(x))$; this means that Y is an efficient set and will therefore complete the proof.

For $i = 1, \ldots, r$, $f_i(x) \leq M_V(I) \leq \overline{M}_V(I)$, so $\varepsilon f_i(x) \leq \overline{M}_V(I) / \left(1 + \overline{M}_V(I)\right) < 1$. Since f_i maps into $\mathcal{Z}^+$, the statement that $D_\omega^{(\varepsilon)}(f(y), f(x))$ becomes

$$\begin{cases} \omega_i = \text{``} \leq \text{''} & \Rightarrow \quad f_i(y) \leq f_i(x) \ , \\ \omega_i = \text{``} \geq \text{''} & \Rightarrow \quad f_i(y) \geq f_i(x) \ , \\ \omega_i = \text{``} = \text{''} & \Rightarrow \quad f_i(y) = f_i(x) \ ; \end{cases}$$

so $D_\omega(f(y), f(x))$. ■

The converse of Theorem 1, however, cannot true unless $\mathcal{P} = \mathcal{NP}$. The following example presents a problem that has a VPP algorithm but no FAS unless $\mathcal{P} = \mathcal{NP}$. Conditions under which the converse is true are developed in Section 4.2 and stated formally in Section 4.3.

Example 9 (Problem with VPP algorithm, but no FAS unless $\mathcal{P} = \mathcal{NP}$) In order to show that the the converse of Theorem 1 is false the following problem will be useful:

Minimum Exact Cover by 3-Sets (Min-X3C)

Instance: $(n, m, A, \mathcal{B}, c)$. The number of triples n and the solution size m are non-negative integers such that $m \leq n$. A is a set of items with $|A| = 3m$. $\mathcal{B}$ is a collection of n triples $B_j \subseteq A$ with $|B_j| = 3$, $j = 1, \ldots, n$. The triple costs c_j, $j = 1, \ldots, n$, are non-negative integers.

Question: If $\mathcal{B}' \subseteq \mathcal{B}$, then the cost of $\mathcal{B}'$ is $\sum_{j | B_j \in \mathcal{B}'} c_j$. An exact cover of A is a set $\mathcal{B}' \subseteq \mathcal{B}$, with $|\mathcal{B}'| = m$, such that $\bigcup_{j | B_j \in \mathcal{B}'} B_j = A$. Find an exact cover of A of minimum cost.

Restricting attention to those instances in which $c_j = 1$, $j = 1, \ldots, n$, yields the problem Exact Cover by 3-Sets (X3C). Since X3C is $\mathcal{NP}$-hard[GJ79], Min-X3C is strongly $\mathcal{NP}$-hard. Therefore no FAS can exist for Min-X3C unless $\mathcal{P} = \mathcal{NP}$[GJ78]. Notice, though, that the number of possible solutions is $\binom{n}{m} \leq 2^n$, so Min-X3C can be solved by enumeration in time $O(n2^n)$. This time bound is not pseudo-polynomial.

Now consider the restriction of Min-X3C to the problem Min-Expensive-X3C, which consists of those instances of Min-X3C in which $c_j \in [2^n, 3^n]$, $j = 1, \ldots, n$, where n is the number of variables in the instance. An instance I of Min-Expensive-X3C has length $L(I) \in \Theta(n \log(m) + n \log(c_{max})) \subseteq \Omega(n^2)$ and largest value $M_V(I) \in \Omega(c_{min}) \subseteq \Omega(2^n)$, so Min-Expensive-X3C can be solved by enumeration in time $O(n2^n) \subseteq O(L(I) \cdot M_V(I))$. In other words, Min-Expensive-X3C has a VPP algorithm. It cannot, however, have an FAS unless $\mathcal{P} = \mathcal{NP}$, because restricting Min-Expensive-X3C to instances in which all costs are 2^n yields the X3C problem. Therefore the converse of Theorem 1 cannot be true unless $\mathcal{P} = \mathcal{NP}$. ■

4.2 Regularity Conditions

Despite the previous example, if a problem satisfies suitable regularity conditions, then the existence of an FAS corresponds exactly to the existence of a VPP algorithm. Of course, Min-Expensive-X3C does not satisfy these conditions.

The conditions will be developed in this section by examining a common way to devise an FAS. In the course of analyzing the algorithm being considered, two difficulties will be encountered that would prevent the algorithm from being an FAS. Simple regularity conditions on a problem that preclude these difficulties from arising will be described. In the next section these conditions will be proved to be sufficient to guarantee the existence of an FAS for a problem.

One way to solve a problem approximately but quickly is to use a fast algorithm to solve a simpler version of the problem. In the present context, "approximately but quickly" means within the accuracy and time bounds that define an FAS. One method that has been used to implement this idea is to use a VPP algorithm to solve a version of the original instance in which some precision is lost in the objective function. In other words, the low-order bits of the objective function values are dropped, thereby creating a new instance that is solved using a VPP algorithm. This approach is called *scaling.*

One effect of scaling is to reduce the magnitude of the largest value of the objective function. Since the time bound of the VPP algorithm typically increases monotonically with this largest value, scaling an instance reduces the time needed to apply the algorithm.

The price of this decreased time requirement, however, is that the wrong instance has been solved. Although solutions to the scaled instance may not exactly solve the original instance, the loss of accuracy may be acceptable because an FAS need only generate approximate solutions. The key is to find a method of scaling that reduces the running time sufficiently without losing too much accuracy, that is, to scale so as to meet the requirements of an FAS.

The remainder of this section shows how to do just that. First the notion of scaling is formalized. The performance of the scaling algorithm is then analyzed to ensure that it is accurate and fast enough to qualify as an FAS. As mentioned above, this analysis will uncover two difficulties that will be circumvented by requiring problems to satisfy regularity conditions.

Scaling. Scaling is simply the loss of low-order precision in objective function values. The definition is given in terms of arbitrary scale factors. Later analysis will use scale factors that are powers of two so that the loss of precision can be counted as a number of bits.

Definition 1 (Function scaling) Let f be an r-criteria function and $t \in \mathcal{Z}^{r+}$ with $t_i > 0$, $i = 1, \ldots, r$. The function f *scaled by* t, written $g = \lfloor f/t \rfloor$, is defined by $g_i(x) = \lfloor f_i(x) / t_i \rfloor$, $i = 1, \ldots, r$. ∎

The following inequalities highlight some effects of scaling. If $v_i \geq 0$ and $t_i > 0$, then

$$v_i - t_i < t_i \lfloor v_i/t_i \rfloor \leq v_i < t_i \lfloor v_i/t_i \rfloor + t_i \ . \tag{2}$$

Choosing an appropriate value for t will be discussed later in this section. First, however, another difficulty must be addressed. The idea described above is to scale the instance and then apply a VPP algorithm. Unfortunately, an algorithm that is VPP using the original objective function need not be VPP using the scaled objective function. To see this, consider scaling an n-variable instance of Min-Expensive-X3C by $t = 2^n e_n$. Although Min-Expensive-X3C has a VPP algorithm, the problem consisting of the scaled instances does not, unless $\mathcal{P} = \mathcal{NP}$.

In order to avoid this problem with scaling, attention in the rest of this paper will be restricted to problems with which this difficulty cannot occur. The following regularity condition on the class of objective functions guarantees that the scaled instances can be solved in VPP time.

Definition 2 (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 $t \in \mathcal{Z}^{r+}$ with $t_i > 0$, $i = 1, \ldots, r$. ■

The following example illustrates this definition.

Example 10 (Closure under scaling) Some examples of function families that *are* closed under scaling are:

1. The class of all weakly monotonically increasing functions. A function $f : S \rightarrow \mathcal{Z}^{r+}$ is in this class if for $x, y \in S$, $x \leq y$ implies that $f(x) \leq f(y)$.
2. The class of all functions that map into $\{0, \ldots, k\}$ for some fixed $k > 0$.

Some examples of function families that *are not* closed under scaling are:

1. The class of all linear functions.
2. The class of all functions that map into $\{1, \ldots, k\}$ for some fixed $k > 0$. The objective function in Min-Expensive-X3C is not closed under scaling for the same reason.
3. The class of all convex functions.

The last fact is seen using the convex function $f(x) = x^2$ and the scale factor $t = 9$. Let $g(x) = \lfloor f(x)/9 \rfloor$. Some values of these functions are illustrated in Figure 6; the round dots represent values of $f(x)$ and the square dots represent values of $g(x)$. The dashed line shows that $g(x)$ is not convex. ■

Accuracy. In order to determine an appropriate value for the vector t of scale factors, consider what happens when a VPP algorithm is used to solve an instance with a scaled objective function. As mentioned above, 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 $p \in \mathcal{Z}^{r+}$, set $t_i = 2^{p_i}$, $i = 1, \ldots, r$, and let $g = \lfloor f/t \rfloor$ be a scaled objective function. Solve the scaled instance $I_t = (S, g, \omega)$ to find a solution set C_t. Now consider how closely any particular value of $f(x)$ is approximated by C_t.

The set C_t must contain a point y such that $D_\omega(g(y), g(x))$. Suppose that for some $i \in \{1, \ldots, r\}$, $\omega_i =$ " $\leq$ "; then

$$g_i(y) \leq g_i(x) \ . \tag{3}$$

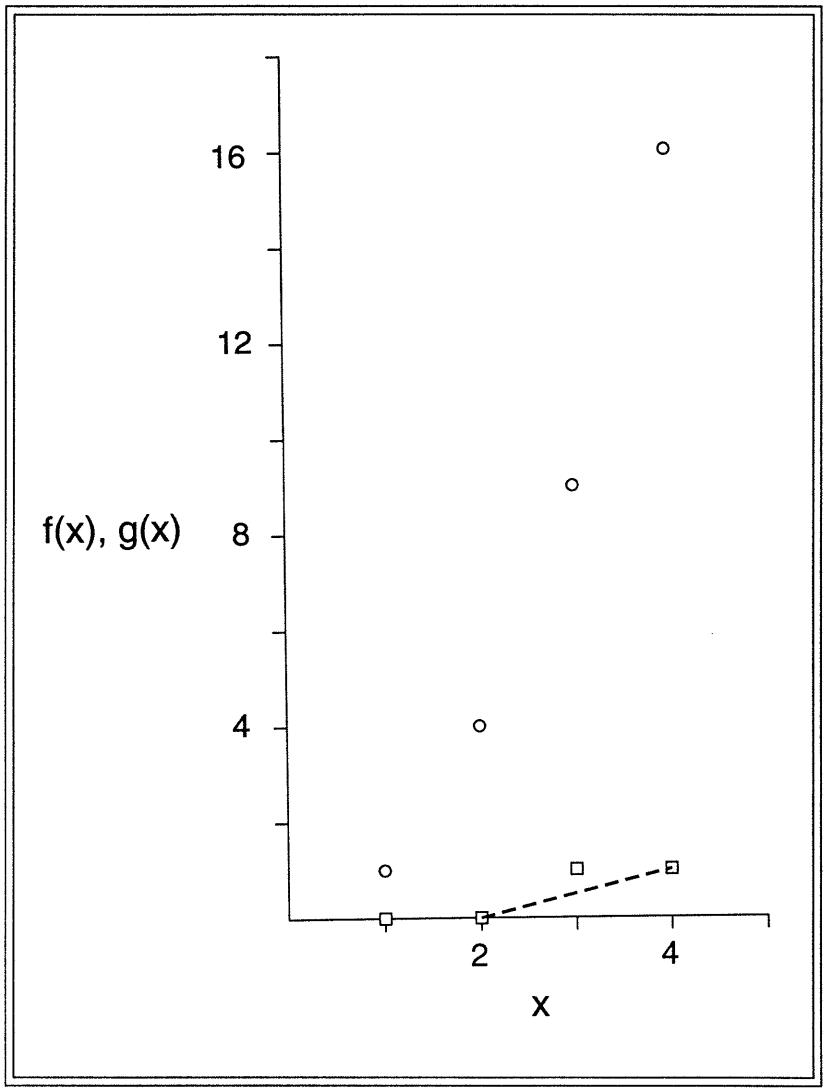

Figure 6: Round dots are values of $f(x) = x^2$ and square dots are values of $g(x) = \lfloor f(x)/9 \rfloor$. The dashed line shows that $g(x)$ is not convex.

In order for C_t to be the output of an FAS, the value of p_i must be chosen so that $f_i(y) \le (1+\varepsilon) f_i(x)$. Substituting for g in Equation 3 and applying Equation 2 yields $f_i(y) < f_i(x) + t_i$, so p_i must be chosen so that $t_i \le \varepsilon f_i(x)$. When $\varepsilon f_i(x) \ge 1$, setting

$$p_i = \lfloor \log(\varepsilon f_i(x)) \rfloor \tag{4}$$

will guarantee that $f_i(y)$ is close enough to $f_i(x)$. When $\varepsilon f_i(x) < 1$, setting $p_i = 0$ works, and the original objective function can be used. 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 as regards this particular value of $f(x)$.

Running Time. The FAS must, however, find a solution for *each* possible value of $f(x)$. This will be achieved by solving a sequence of scaled instances, each with a distinct scaling vector t, so that an appropriate scaling vector is used for each possible value of $f(x)$. This means that instances must be solved with $p_i = 0, \ldots, p^*$, where $p^* = \max\left\{ \left\lfloor \log\left(\varepsilon \overline{M}_{\mathrm{V}}(I)\right) \right\rfloor, 0 \right\}$.

The problem with solving so many scaled instances is the time required for the computations. A simple upper bound on the time needed is the product of the number η of instances that must be solved and the maximum time τ needed to solve a single scaled instance.

The number of instances is $\eta = (1+p^*)^r$. If $\varepsilon \overline{M}_{\mathrm{V}}(I) < 1$, then $\eta = 1$. Otherwise,

$$\begin{aligned} \eta &\le \left(1 + \log\left(\varepsilon \overline{M}_{\mathrm{V}}(I)\right)\right)^r \\ &\le \left(1 + \log(\varepsilon) + \log\left(\overline{M}_{\mathrm{V}}(I)\right)\right)^r . \end{aligned}$$

The value of η grows with the value of ε, that is, as the required accuracy decreases. If ε is too large, i.e., if not much accuracy is required, then too much time will be required for this approach to be an FAS. The analysis so far will, however, remain valid if greater accuracy is used. Therefore if $\varepsilon > 2$, it will be replaced by $\varepsilon' = 2$ in order to bound the running time. With this modification to the solution scheme,

$$\begin{aligned} \eta &\le \left(2 + \log\left(\overline{M}_{\mathrm{V}}(I)\right)\right)^r \\ &\in O\left([L(I)]^{k_1}\right) , \end{aligned}$$

for some $k_1 \in \mathcal{Z}^+$, because $\log(M_{\mathrm{V}}(I)) \le L(I)$ and r is fixed.

All that remains is to bound τ, the time needed to solve a single scaled instance I_t. For starters, $\tau \in O\left([L(I_t) \cdot M_{\mathrm{V}}(I_t)]^{k_2}\right)$, for some $k_2 \in \mathcal{Z}^+$, because the algorithm used to solve I_t is VPP. If $M_{\mathrm{V}}(I_t)$ is large, however, then τ will be large as well. But instances with large values of $M_{\mathrm{V}}(I_t)$ are superfluous: for any

$x \in S$, define t as described above, and then

$$\begin{aligned} g_i(x) &= \lfloor f_i(x)/t_i \rfloor \\ &\leq f_i(x)/t_i \\ &= f_i(x)/2^{\lfloor \log(\varepsilon' f_i(x)) \rfloor} \\ &< 2/\varepsilon' \\ &< \lfloor 4/\varepsilon' \rfloor . \end{aligned}$$

The final inequality is true because $\varepsilon' \leq 2$ implies that $\frac{4}{\varepsilon'} - \frac{2}{\varepsilon'} \geq 1$. In other words, scaling to attain a relative accuracy of ε allows attention to be restricted to objective function values that do not exceed $\lfloor 4/\varepsilon \rfloor$.

It is sufficient to consider only those instances I_t in which $M_V(I_t) \leq \lfloor \frac{4}{\varepsilon'} \rfloor = \mu^*$. If the largest value of each other instance is set to μ^*, the unneeded instances will effectively be automatically discarded. The limits on the values of the objective function values are called *box constraints.*

Definition 3 (Box constraints) Let f be an r-criteria function and $M \in \mathcal{Z}^{r+}$. The function *f with box constraints M*, written $g = \min\{f, M\}$, is defined by $g_i(x) = \min\{f_i(x), M_i\}$, $i = 1, \ldots, r$. ■

Assumption 2 ensures that a box-constrained function can be computed efficiently, and the discussion following that assumption explains why the assumption can be made essentially without loss of generality.

An algorithm that is VPP for a problem with box constraints requires less running time than it does for the unconstrained problem if the unconstrained problem has function values larger than the bound. If the box constraints constraints are not tight, then the time bounds are the same. The box constraints are used to avoid spending too much time solving any particular instance. The second regularity condition on the class of objective functions guarantees that each box-constrained instance can be solved quickly enough for the entire scheme to fit the running time requirements of an FAS.

Definition 4 (Closure under box constraints) 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 $M \in \mathcal{Z}^{r+}$, $\min\{f, M\} \in \mathcal{F}$. ■

By further restricting attention to objective functions that are closed under box constraints as well as under scaling, the time τ required to solve a particular instance can be no greater than $O\left([L(I_t)/\varepsilon']^{k_3}\right)$, for some $k_3 \in \mathcal{Z}^+$. The total time needed to solve all the scaled instances is therefore bounded by $\eta\tau \in O\left([L(I)/\varepsilon']^{k}\right)$, for some $k \in \mathcal{Z}^+$. Since $\varepsilon' = \varepsilon$ or $\varepsilon' = 2$, the FAS running time requirements are satisfied.

Separable Objective Functions. As will be seen in [SO94], certain classes of functions that are commonly encountered in practice are closed neither under scaling nor under box constraints. In particular, the class of additively separable functions is not closed under either of these operations. Modifying the closure conditions to accommodate separable objective functions would be appropriate because they are so common. Each term of a separable function must satisfy the two previous closure conditions, so the modified conditions must account for the error that accumulates in the sum of n terms. The following weaker conditions capture the behavior of separable functions.

Definition 5 (Quasi-closure under scaling) Let $\mathcal{F}$ be a family of r-criteria functions. Suppose that for any $f \in \mathcal{F}$ and any $t \in \mathcal{Z}^{r+}$ with $t_i > 0$ for $i = 1, \ldots, r$, there is a $g \in \mathcal{F}$ such that for any $x \in \mathcal{Z}^{n+}$ and $i = 1, \ldots, r$,

if $t_i = 1$, then $\quad g_i(x) = f_i(x)$;

if $t_i > 1$, then $\quad \lfloor f_i(x)/t_i \rfloor - n + 1 \le g_i(x) \le \lfloor f_i(x)/t_i \rfloor$.

Then the family $\mathcal{F}$ is said to be *quasi-closed under scaling* and the function g is called a *scaling neighbor* of f w.r.t. t. ∎

Definition 6 (Quasi-closure under box constraints) Let $\mathcal{F}$ be a family of r-criteria functions. Suppose that for any $f \in \mathcal{F}$ and any $M \in \mathcal{Z}^{r+}$, there is a $g \in \mathcal{F}$ such that for any $x \in \mathcal{Z}^{n+}$ and $i = 1, \ldots, r$,

if $f_i(x) \le M_i$, then $\quad g_i(x) = f_i(x)$;

if $f_i(x) > M_i$, then $\quad M_i \le g_i(x) \le \min\{f_i(x), nM_i\}$.

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 w.r.t. M. ∎

A function class that is closed under scaling is also quasi-closed under scaling, since in that case $\lfloor f/t \rfloor$ is a scaling neighbor of f w.r.t. t. A function class that is closed under box constraints is also quasi-closed under box constraints, since in that case $\min\{f, M\}$ is a box constraint neighbor of f w.r.t. M.

The theorem about the sufficient condition is stated using the quasi-closure regularity conditions so that it applies to problems that have separable objective functions. In addition, the error tolerance is reduced by a factor of n so that any loss of accuracy can be spread among the n terms of the objective function.

4.3 Sufficient Condition

The sufficiency condition is stated and proved in this section. The theorem is the converse of Theorem 1 except for the quasi-closure conditions on $\mathcal{F}$. It is primarily useful if a scaling neighbor and a box constraint neighbor of a function are reasonably easy to find for each function in $\mathcal{F}$, though it is of interest even if they are not. As discussed in [SO94], however, such neighbors are often conveniently available.

The proof follows the development of the previous section. It shows that the existence of a VPP algorithm for a problem is sufficient for the existence of an FAS for the problem by using the VPP algorithm to define the FAS. Because of the generality of the method, however, the FAS so derived may not be the most efficient approximation scheme possible. Typically, an FAS constructed as described in this proof can be improved by adapting it to the special characteristics of the problem.

Theorem 2 (Sufficient conditions for existence of an FAS) *Consider an optimization problem* $\Pi = (\Psi, \mathcal{F}, \omega, Opt)$, *where* $\mathcal{F}$ *is a class of* r*-criteria functions that is quasi-closed both under scaling and under box constraints. If* Π *has a VPP algorithm, then* Π *has an FAS.*

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

$$\begin{aligned}
\varepsilon' &= \tfrac{1}{n}\min\{\varepsilon, 2\}\ ,\\
\mu &= \lfloor 4/\varepsilon' \rfloor\, e_r \quad \text{(recall that } e_r \text{ is the } r\text{-vector of ones)},\\
p^* &= \begin{cases} \lfloor \log(\varepsilon' \overline{M}_V(I)) \rfloor & \text{if } \varepsilon' \overline{M}_V(I) \geq 1\,,\\ 0 & \text{otherwise,}\end{cases}\\
T &= \{t \in \mathcal{Z}^{r+} : \exists\, p \in \{0, \ldots, p^*\}^r \ni t_i = 2^{p_i},\ i = 1, \ldots, r\}\ .
\end{aligned}$$

So T is the set of all r-vectors whose components are integral powers of two up to $\varepsilon' \overline{M}_V(I)$. The following algorithm will be shown to be an FAS for Π:

1. Determine $\overline{M}_V(I)$ satisfying Equation 1. Set $C \leftarrow \emptyset$.
2. For each $t \in T$, let g be a scaling neighbor of f w.r.t. t and let h be a box constraint neighbor of g w.r.t. μ. Use $\mathcal{A}$ to solve the instance $I_t = (S, h, \omega)$, 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.

The algorithm is first shown to run within the time allowed for an FAS, then its correctness is demonstrated.

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

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

$$\begin{aligned}
\eta &< \left(1 + \log(\varepsilon' \overline{M}_V(I))\right)^r\\
&\leq \left(2 - \log(n) + \log(\overline{M}_V(I))\right)^r\\
&\in O\left([L(I)]^{k_1}\right),
\end{aligned}$$

for some $k_1 \in \mathcal{Z}^+$, since $\varepsilon' \leq 2/n$, $\log(M_V(I)) \leq L(I)$, and r is fixed.

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

$$\begin{aligned}
\tau &\in O\left([L(I_t) \cdot M_V(I_t)]^{k_2}\right)\\
&\subseteq O\left([L(I_t) \cdot ||\mu||]^{k_2}\right)\\
&= O\left([L(I)/\varepsilon']^{k_3}\right),
\end{aligned}$$

for some $k_2, k_3 \in \mathcal{Z}^+$. So the total time needed by the algorithm is at most, for some $k \in \mathcal{Z}^+$, $\eta\tau \in O\left([L(I)/\varepsilon']^k\right)$. But $\varepsilon' = \varepsilon/n$ or $\varepsilon' = 2/n$, so the algorithm runs within the time allowed for an FAS.

Accuracy: In order for the algorithm to be correct, it must be the case that for each $x \in S$, there is some

$y \in C'$ such that $D_\omega^{(\varepsilon)}(f(y), f(x))$. For any $t \in T$, the function g satisfies, for $x \in S$:

$$\text{if } t_i = 1, \quad \text{then} \quad g_i(x) = f_i(x) \ , \tag{5a}$$

$$\text{if } t_i > 1, \quad \text{then} \quad \lfloor f_i(x)/t_i \rfloor - n + 1 \leq g_i(x) \leq \lfloor f_i(x)/t_i \rfloor \ ; \tag{5b}$$

and the function h satisfies

$$\text{if } g_i(x) \leq \mu_i, \quad \text{then} \quad h_i(x) = g_i(x) \ , \tag{6a}$$

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

Define t as follows: for $i = 1, \ldots, r$,

$$t_i = \begin{cases} 2^{\lfloor \log(\varepsilon' f_i(x)) \rfloor} & \text{if } \varepsilon' f_i(x) \geq 1, \\ 1 & \text{otherwise.} \end{cases}$$

In particular, if $\varepsilon' f_i(x) \geq 1$, then the following relationships hold, as in Equation 2:

$$\varepsilon' f_i(x)/2 < t_i \leq \varepsilon' f_i(x) \ . \tag{7}$$

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 $i = 1, \ldots, r$,

$$\begin{aligned} \lfloor f_i(x)/t_i \rfloor &\leq f_i(x)/t_i \\ &< 2/\varepsilon' \\ &< \lfloor 4/\varepsilon' \rfloor \\ &= \mu_i \ . \end{aligned} \tag{8}$$

The first strict inequality, which is straightforward when $\varepsilon' f_i(x) < 1$, follows from Equation 7 otherwise. Equation 8 shows that the box constraints μ are not tight at x when f is scaled by t. Combining Equations 5 and 8 yields

$$g_i(x) < \mu_i \ , \tag{9}$$

so by Equation 6a,

$$h_i(x) = g_i(x) \ , \tag{10}$$

and substituting into Equations 5,

$$\lfloor f_i(x)/t_i \rfloor - n + 1 \leq h_i(x) \leq \lfloor f_i(x)/t_i \rfloor \ . \tag{11}$$

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, depending on x, that satisfies $D_\omega\left(h(y), h(x)\right)$. It will be shown that y also satisfies $D_\omega^{(\varepsilon')}\left(f(y), f(x)\right)$. Since $\varepsilon' \leq \varepsilon$, this will complete the proof.

Consider any index $i \in \{1, \ldots, r\}$. Assume that $\omega_i =$ "$\leq$"; the proofs for $\omega_i \in \{$"$\geq$", "$=$"$\}$ are similar. So

$$h_i(y) \leq h_i(x) \; ; \tag{12}$$

it must be shown that $f_i(y) \leq (1+\varepsilon') f_i(x)$. By Equations 12, 10, and 9,

$$\begin{aligned} h_i(y) &\leq h_i(x) \\ &= g_i(x) \\ &< \mu_i \, , \end{aligned}$$

which means, because of Equation 6b, that $g_i(y) \leq \mu_i$. By Equation 6a,

$$h_i(y) = g_i(y) \, , \tag{13}$$

and substituting into Equations 5,

$$\lfloor f_i(y)/t_i \rfloor - n + 1 \leq h_i(y) \leq \lfloor f_i(y)/t_i \rfloor \, . \tag{14}$$

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

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

$$\begin{aligned} \lfloor f_i(y)/t_i \rfloor &\leq h_i(y) + n - 1 && \text{by Equation 14} \\ &\leq h_i(x) + n - 1 && \text{by Equation 12} \\ &\leq \lfloor f_i(x)/t_i \rfloor + n - 1 && \text{by Equation 11} \, , \end{aligned}$$

and so

$$\begin{aligned} f_i(y) &< t_i \lfloor f_i(y)/t_i \rfloor + t_i && \text{by Equation 2} \\ &\leq t_i \lfloor f_i(x)/t_i \rfloor + n t_i && \\ &\leq f_i(x) + n t_i && \text{by Equation 2} \\ &\leq (1 + n\varepsilon') f_i(x) && \text{by Equation 7} \\ &\leq (1+\varepsilon) f_i(x) \, . && \end{aligned}$$

Case 2: $\varepsilon' f_i(x) < 1$. Then $t_i = 1$, and

$$\begin{array}{rcll} f_i(y) & = & g_i(y) & \text{by Equation 5a} \\ & = & h_i(y) & \text{by Equation 13} \\ & \leq & h_i(x) & \text{by Equation 12} \\ & = & g_i(x) & \text{by Equation 10} \\ & = & f_i(x) & \text{by Equation 5a} \\ & < & (1+\varepsilon') f_i(x) \ . & \end{array}$$

■

In summary, then, an FAS 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 result highlights the key aspects of the sufficiency condition presented in [PS82]. The next section shows that this result is stronger than the result in [PS82] when problems with general integer variables are considered.

4.4 Non-Binary Domains

This section highlights the case of non-binary domains, that is, problems in which the variables can take values other than zero or one. Much of the literature on approximation schemes only considers binary domains, that is, 0-1 problems[HS74, Bab75, GL78, GL79a]. The exceptions, however, generally treat non-binary problems by reduction to the binary case[IK75, Law79, KK84].

The idea behind the typical reduction from a general integer problem to a related binary problem is to consider the binary representation of the general integer variables. A separate binary variable is used to represent each bit of the general integer variables. This technique is demonstrated in Example 11. It only works, however, for *separable linear* objective functions, not for the more general objective functions considered in this paper. The same technique can be applied to the unary expansion of each general integer variable, but this approach generally yields a pseudo-polynomial time algorithm, not a VPP algorithm.

The algorithm in the sufficiency condition of [PS82], rather than being VPP, runs in time $O\left([L(I) \cdot p_{max}]^{k_1}\right)$, for some $k_1 \in \mathcal{Z}^+$. For binary domains, this is essentially the same as a VPP algorithm. For general integer domains, however, Theorem 2 is stronger; it guarantees the existence of an FAS even if the optimization algorithm uses as much as time as $O\left([L(I) \cdot p_{max} \cdot u_{max}]^{k_2}\right)$, for some $k_2 \in \mathcal{Z}^+$. This stronger result derives from having identified the sufficient conditions for the existence of an FAS more precisely than has been done before.

Example 11 (Reducing integer variables to binary variables) A typical reduction from general integer variables to binary variables is illustrated using a variant of the Max Binary Knapsack problem defined in Section 2.2. In this version of the problem the variables can take general non-negative integer values.

Max Integer Knapsack

Instance: (n, p, V, v, u). The number of items n; the profits p_j, $j = 1, \ldots, n$; the knapsack volume V; the item volumes v_j, $j = 1, \ldots, n$; and the item quantity upper bounds u_j, $j = 1, \ldots, n$; are non-negative integers such that $v_j \leq V$, $j = 1, \ldots, n$.

Question: Find x to solve

$$\begin{aligned} \max \sum_{j=1}^{n} & \; p_j x_j \\ \text{s.t.} \sum_{j=1}^{n} & \; v_j x_j \leq V \\ & \; x_j \in \{0, \ldots, u_j\}, \; j = 1, \ldots, n \end{aligned}$$

Given an instance $I = (n, V, p, v, u)$ of this problem, let $l_j = 1 + \lfloor \log(u_j) \rfloor$ be the number of bits in the binary expansion of u_j, $j = 1, \ldots, n$. An instance of the Max Binary Knapsack problem which contains a variable corresponding to each such bit will be constructed.

The variables of the Max Binary Knapsack instance are y_{jk}, $j = 1, \ldots, n$, $k = 1, \ldots, l_j$. The variable y_{jk} represents the k^{th} low-order bit of the binary expansion of x_j. A solution of this instance with $y_{jk} = \overline{y}_{jk}$ will correspond to a solution of the Max Integer Knapsack instance with $x_j = \sum_{k=1}^{l_j} 2^{k-1} \overline{y}_{jk}$. To accomplish this, set

$$\begin{aligned} n' &= \sum_{j=1}^{n} l_j, \\ p'_{jk} &= 2^{k-1} p_j, \quad j = 1, \ldots, n, \quad k = 1, \ldots, l_j, \\ v'_{jk} &= 2^{k-1} v_j, \quad j = 1, \ldots, n, \quad k = 1, \ldots, l_j. \end{aligned}$$

Now construct and solve the instance $I' = (n', V, p', v')$ of Max Binary Knapsack. The instances I' and I have the same optimal value, and setting

$$x_j = \sum_{i=1}^{l_j} 2^{k-1} y_{jk}, \; j = 1, \ldots, n,$$

yields a solution to I with the same value as the solution found for I'. ■

Other reductions of problems with general integer variables to problems with binary variables typically use a scheme similar to the one presented in this example.

5 VPP Reductions

Using Theorem 2 to prove the existence of an FAS for each problem of interest would be rather tedious. An alternative is to use *reduction*, a general technique for proving theorems about the ease or difficulty of solving particular problems[GJ79, HU79].

A new kind of reduction that is appropriate for use in proofs about the existence of VPP algorithms is introduced in Section 5.1. Several lemmas that can be used to prove the existence of an FAS are then proved in Section 5.2. Section 5.3 shows how to use reduction to prove that problems are hard.

5.1 Definitions

VPP reductions are defined in this section. Informally, a problem Π VPP reduces to a problem Π' if Π can be solved efficiently by calling a subroutine that solves Π' efficiently. More precisely, a problem Π *VPP reduces* to a problem Π', written $\Pi \propto_{\text{VPP}} \Pi'$, if there is an algorithm $\mathcal{A}$ for Π that uses as a subroutine a (possibly hypothetical) 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 $\Pi \propto_{\text{VPP}} \Pi'$ and $\Pi' \propto_{\text{VPP}} \Pi$, then Π and Π' are *VPP equivalent*, which is written $\Pi \equiv_{\text{VPP}} \Pi'$.

As an alternative, VPP reducibility could be defined using an oracle for $\mathcal{A}'$. Note that the definition is of a reduction, not a transformation; that is, $\mathcal{A}'$ can be called several times during the execution of $\mathcal{A}$, not just once at the very end. Transformations are commonly used in reasoning about the difficulty of recognition problems, and the "yes" or "no" answer generated by the single call to $\mathcal{A}'$ is the answer returned by $\mathcal{A}$. The more general concept of reduction is needed in the present setting, however, since a set of points must be generated, rather than just a single "yes" or "no".

5.2 Using VPP Reductions

This section shows how to use VPP reductions to clarify the difficulty of solving specific problems. They are used in [SO94] to simplify proofs that certain problems have FAS's, and they can also be used to show that problems are difficult. The first three lemmas and the corollary are about the relationship between problems with similar representations. The last lemma is more general, and is used often in [SO94].

The following lemma formalizes the fact that restricting the family of functions cannot lead to a problem that is substantially more difficult to solve.

Lemma 1 *Let Ψ be a family of feasible sets and $\omega \in \Omega^r$. Suppose that $\mathcal{F}$ and $\mathcal{F}'$ are families of r-criteria functions such that $\mathcal{F} \subseteq \mathcal{F}'$. Then $(\Psi, \mathcal{F}, \omega, Feas) \propto_{\text{VPP}} (\Psi, \mathcal{F}', \omega, Feas)$.*

Proof. Let $\mathcal{A}'$ be an algorithm for $(\Psi, \mathcal{F}', \omega, \text{Feas})$. If $I \in (\Psi, \mathcal{F}, \omega, \text{Feas})$, then $I \in (\Psi, \mathcal{F}', \omega, \text{Feas})$ also, since $\mathcal{F} \subseteq \mathcal{F}'$. So a VPP reduction consists of calling $\mathcal{A}'$ to solve I; the solution is the output from $\mathcal{A}'$. ■

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 2 *Let Ψ be a family of feasible sets, $\mathcal{F}$ a family of r-criteria functions that is quasi-closed under box constraints, and $\omega \in \Omega^r$. Then $(\Psi, \mathcal{F}, \omega, \mathit{Feas}) \propto_{\mathrm{VPP}} (\Psi, \mathcal{F}, \omega^{r,=}, \mathit{Feas})$.*

Proof. Given $I = (S, f, \omega, M) \in (\Psi, \mathcal{F}, \omega, \mathrm{Feas})$, let $g \in \Psi$ be a box constraint neighbor of f w.r.t. M. So for all $x \in S$, $g(x) \leq nM$.

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

A feasibility problem is often 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, which shows that an optimization problem is not much more difficult to solve than the corresponding feasibility problem.

Lemma 3 *Let Ψ be a family of feasible sets, $\mathcal{F}$ a family of r-criteria functions that is quasi-closed under box constraints, and $\omega \in \Omega^r$. Then $(\Psi, \mathcal{F}, \omega, \mathit{Feas}) \equiv_{\mathrm{VPP}} (\Psi, \mathcal{F}, \omega, \mathit{Opt})$.*

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

$\underline{\Pi_F \propto_{\mathrm{VPP}} \Pi_O:}$ Let $I_F = (S, f, \omega, M) \in \Pi_F$ and $g \in \Psi$ be a box constraint neighbor of f w.r.t. M; so $I_O = (S, g, \omega) \in \Pi_O$. A solution set C_O for I_O contains at most $(n\,\|M\| + 1)^r$ 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.

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

$$H = \{\, M \in \mathcal{Z}^{r+} : \|M\| \leq \overline{M}_{\mathrm{V}}(I_O) \,\};$$

so $|H| = \left(\overline{M}_{\mathrm{V}}(I_O) + 1\right)^r$. For each target vector $M \in H$, let $C_F(M)$ be the solution to $(S, f, \omega, M) \in \Pi_F$, and define $C_F = \bigcup_{M \in H} C_F(M)$. The solution to I_O is a subset of C_F. A VPP reduction consists of solving (S, f, ω, M) for each $M \in H$ and eliminating unneeded elements from C_F. ∎

The utility of Lemma 3 can be seen by applying it to Lemmas 1 and 2 to obtain the following corollary.

Corollary 1 *Let Ψ be a family of feasible sets, $\mathcal{F}$ and $\mathcal{F}'$ families of r-criteria functions that are quasi-closed under box constraints and satisfy $\mathcal{F} \subseteq \mathcal{F}'$, and $\omega \in \Omega^r$. Then $(\Psi, \mathcal{F}, \omega, \mathit{Opt}) \propto_{\mathrm{VPP}} (\Psi, \mathcal{F}', \omega, \mathit{Opt})$ and $(\Psi, \mathcal{F}, \omega, \mathit{Opt}) \propto_{\mathrm{VPP}} (\Psi, \mathcal{F}, \omega^{r,=}, \mathit{Opt})$.*

The following lemma is used repeatedly in later sections. Note that parts of it are true with weaker assumptions about the family $\mathcal{F}$ of objective functions.

Lemma 4 *Let Ψ be a family of feasible sets, $\mathcal{F}$ a family of r-criteria functions that is quasi-closed under scaling and under box constraints, and $\omega \in \Omega^r$. Let Π' be a feasibility or optimization problem that has a VPP algorithm or an optimization problem that has an FAS. If $(\Psi, \mathcal{F}, \omega, \textit{Feas}) \propto_{\text{VPP}} \Pi'$ or $(\Psi, \mathcal{F}, \omega, \textit{Opt}) \propto_{\text{VPP}} \Pi'$, then $(\Psi, \mathcal{F}, \omega, \textit{Opt})$ has an FAS.*

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

If $(\Psi, \mathcal{F}, \omega, \text{Feas}) \propto_{\text{VPP}} \Pi'$, then by Lemma 3, $(\Psi, \mathcal{F}, \omega, \text{Opt}) \propto_{\text{VPP}} \Pi'$ as well; so under either reduction assumption, $(\Psi, \mathcal{F}, \omega, \text{Opt})$ has a VPP algorithm. Theorem 2, then, implies that $(\Psi, \mathcal{F}, \omega, \text{Opt})$ has an FAS.
■

5.3 Proving that Problems are Hard

This section shows one way of using VPP reductions to prove that problems are hard. 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.

The method used in this paper to specify instances and problems is not the only possible approach. An alternative formulation might describe an instance $I \in \Pi$ as a string of characters from some finite set, say $\{0,1\}$. In this alternative formulation, the instance length $L(I)$ would naturally be defined as the number of characters in this string. The largest value $M_{\text{V}}(I)$, however, would be harder to identify in general; it would depend more specifically on the problem. Defining the largest value would require imposing a structure on the string, so that the "values," each of which is defined by some subset of characters, can be recognized within the context of the entire string. This issue arises, for example, when defining the largest integer in an instance in discussions of strong $\mathcal{NP}$-hardness[GJ79, Section 4.2.1].

In general, let β be the set of integers of interest for instances of a problem Π. In a discussion about strong $\mathcal{NP}$-hardness, for example, the set of integers that appear in the problem statement is important, and this set would be β. In a discussion of ordinary $\mathcal{NP}$-hardness, however, particular integers are not identified, so $\beta = \emptyset$. In the present setting, an appropriate choice for β will be seen to be V, the set of possible objective function values.

Definition 7 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 8 A problem Π is β-strongly NP-hard if, for some polynomial $p(\cdot)$, the problem $\Pi_{\beta,p}$ is $\mathcal{NP}$-hard.
■

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

Recall that, if $\mathcal{P} \neq \mathcal{NP}$, then no $\mathcal{NP}$-hard problem has a polynomial time algorithm, and no strongly $\mathcal{NP}$-hard problem has a pseudo-polynomial time algorithm[GJ78]. In the same vein, the following theorem shows that the appropriate choice for the present context is, as asserted above, $\beta = \mathrm{V}$.

Theorem 3 *If a problem Π is V-strongly NP-hard, then Π has no VPP algorithm unless $\mathcal{P} = \mathcal{NP}$.*

Proof. Since Π is V-strongly NP-hard, there is a polynomial $p(\cdot)$ such that $\Pi_{V,p}$ is $\mathcal{NP}$-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 $M_{\mathrm{V}}(I) \in O\left([L(I)]^k\right)$, for some $k \in \mathcal{Z}^+$. But $\Pi_{V,p}$ is $\mathcal{NP}$-hard, so such an algorithm $\mathcal{A}$ can exist only if $\mathcal{P} = \mathcal{NP}$. ∎

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, as will be shown by Theorem 4. The following lemma, which shows that the magnitudes of the values that occur in constructed instances are not too large, will be useful.

Lemma 5 *Let Π and Π' be problems such that $\Pi \propto_{\mathrm{VPP}} \Pi'$, 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 $I' \in \Pi'$. Then the largest value of I' is polynomial in both the length and largest value of I, that is, $M_{\mathrm{V}}(I') \in O\left([L(I) \cdot M_{\mathrm{V}}(I)]^k\right)$, for some $k \in \mathcal{Z}^+$.*

Proof. Suppose that this inequality does not hold, but that $\mathcal{A}'$ is a VPP algorithm for Π'. If $M_{\mathrm{V}}(I')$ appears in a tight time bound for $\mathcal{A}'$ with a positive coefficient, then the time for a single call to $\mathcal{A}'$, and hence the time for $\mathcal{A}$, will be super-polynomial in $L(I)$, $M_{\mathrm{V}}(I)$, or both. Therefore $\mathcal{A}$ will not be a VPP algorithm for Π, contradicting the assumption that $\Pi \propto_{\mathrm{VPP}} \Pi'$. ∎

Now to the point. The following theorem parallels the $\mathcal{NP}$-hardness results mentioned earlier and can be used in the same ways as those theorems to show that problems are difficult.

Theorem 4 *Let Π and Π' be problems. If Π is V-strongly NP-hard and $\Pi \propto_{\mathrm{VPP}} \Pi'$, 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 $\mathcal{NP}$-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

$$\begin{aligned} M_{\mathrm{V}}(I') &\leq p_2(L(I) \cdot M_{\mathrm{V}}(I)) && \text{by Lemma 5} \\ &\leq p_2(L(I) \cdot p_1(L(I))) && \text{because } I \in \Pi_{V,p_1} \\ &\leq p_3(L(I)) \ . \end{aligned}$$

Assume for the moment that $L(I) \in O\left([L(I')]^k\right)$, for some $k \in \mathcal{Z}^+$. 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 $\Pi_{V,p_1} \propto_{\text{VPP}} \Pi'_{V,p_4}$.

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 $\Pi_{V,p_1} \propto_{\text{T}} \Pi'_{V,p_4}$, where $\propto_{\text{T}}$ denotes polynomial time Turing reducibility[GJ79, section 5.1]. But Π_{V,p_1} is $\mathcal{NP}$-hard, so Π'_{V,p_4} is also $\mathcal{NP}$-hard, and hence Π' is V-strongly NP-hard.

This proof depends on the assumption that $L(I) \in O\left([L(I')]^k\right)$, for some $k \in \mathcal{Z}^+$; 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 condition is true, so the assumption can be made without loss of generality. ■

Padding instance descriptions with blanks violates the convention that instances should be described concisely. Allowing such descriptions, that is, including more instances in a problem, does not make the problems under consideration any easier. It also does not make them much harder, and so does not have a major effect on relevant complexity results. In addition, reductions used in practice rarely, if ever, need such padding.

6 Summary

This paper has introduced a new framework in which to consider the optimization of multi-criteria combinatorial optimization problems. Finding the entire efficient frontier can be quite difficult, so the approach taken here has been to approximate it; this is in contrast to previous work in which parts of exact solutions were found.

Necessary and sufficient conditions for the existence of a fast approximation scheme for a problem were motivated and proved. Besides extending previously known results from the single-criterion setting to problems with multiple criteria, these conditions, when restricted to the single criterion case, are more specific than are the previously known results.

These 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.

A companion paper applies these results to a variety of network flow and other combinatorial optimization problems.

Acknowledgments

It is a pleasure to acknowledge and thank our colleagues for their assistance with this research. Tom Magnanti and Cliff Stein provided stimulating discussions and comments. The following people helped us locate relevant previous work: Steve Graves, Toshihide Ibaraki, David Johnson, Francesco Maffioli, Tom McCormick, Alexander Rinnooy Kan, Paul Tseng, and a host of librarians. Special thanks are due to Hitendra Wadhwa for reading and commenting on an earlier version of this manuscript.

This research was supported in part by NSF Presidential Young Investigator grants to James B. Orlin and Shafi Goldwasser, as well as by Air Force Office of Scientific Research grant AFOSR-88-0088 and NSF contract DDM-8921835.

References

[AMOT88] Ravindra K. Ahuja, Kurt Mehlhorn, James B. Orlin, and Robert E. Tarjan. Faster algorithms for the shortest path problem. Technical Report 193, Opns. Res. Center, MIT, April 1988. Published in JACM 1990.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1993.

[AN79] Y. P. Aneja and K. P. K. Nair. Bicriteria transportation problem. *Management Sci.*, 25(1):73–78, January 1979.

[Bab75] L. G. Babat. Linear functionals on the N-dimensional unit cube. *Soviet Mathematics—Doklady*, 16(2):398–400, March 1975. Translated from Russian.

[Bab78] L. G. Babat. A problem with fixed payments. *Engineering Cybernetics*, 16(3):21–27, May 1978. Translated from Russian.

[Bit77] Gabriel R. Bitran. Linear multiple objective programs with zero-one variables. *Math. Prog.*, 13:121–139, 1977.

[Bit79] Gabriel R. Bitran. Theory and algorithms for linear multiple objective programs with zero-one variables. *Math. Prog.*, 17:362–390, 1979.

[BS65] T. A. Brown and R. E. Strauch. Dynamic programming in multiplicative lattices. *J. Math. Anal. and Appl.*, 12(2):364–370, 1965.

[CGM89] P. M. Camerini, G. Galbiati, and F. Maffioli. Combinatorial optimization through algorithms for exact problems and scaling. Rapporto Interno 89-031, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1989.

[CHW76] Ashok K. Chandra, D. S. Hirschberg, and C. K. Wong. Approximate algorithms for some generalized knapsack problems. *Theor. Comp. Sci.*, 3(3):293–304, December 1976.

[CFN77] Gérard Cornuéjols, Marshall L. Fisher, and George L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Sci.*, 23(8):789–810, April 1977.

[DdK80] H. G. Daellenbach and C. A. de Kluyver. Note on multiple objective dynamic programming. *J. Opnl. Res. Soc.*, 31(7):591–594, July 1980.

[Den82] Eric V. Denardo. *Dynamic Programming: Models and Applications.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[DW83] M. E. Dyer and J. Walker. A note on bicriterion programming. *Math. Prog.*, 27(3):355–361, December 1983.

[EP88] V. A. Emelichev and V. A. Perepelitsa. Multiobjective problems on the spanning trees of a graph. *Soviet Mathematics—Doklady*, 37(1):114–117, 1988.

[FG80] Günter Fandel and Tomas Gal. *Multiple Criteria Decision Making—Theory and Application*, volume 177 of *Lecture Notes in Economics and Mathematical Systems.* Springer-Verlag, New York, 1980. Proceedings of the Third Conference, Hagen/Königswinter, West Germany, 20–24 August 1979.

[Fel68] William Feller. *An Introduction to Probability Theory and Its Applications*, volume I. John Wiley & Sons, New York, third edition, 1968.

[Fin75] Yu. Yu. Finkel'shtein. Approximate solution of problems in integral linear programming. *Cybernetics*, 11(3):503–505, May 1975. Translated from Russian. Published July 1976.

[Fin77] Yu. Yu. Finkel'shtein. The ε-approach to the multidimensional knapsack problem: The polynomial growth of the tree of branching. *USSR Computational Mathematics and Mathematical Physics*, 17(4):215–217, 1977. Translated from Russian. Published September 1978.

[Fis80] Marshall L. Fisher. Worst-case analysis of heuristic algorithms. *Management Sci.*, 26(1):1–17, January 1980.

[FBR89] B. Fruhwirth, R. E. Burkard, and G. Rote. Approximation of convex curves with application to the bicriterial minimum cost flow problem. *Eur. J. Opnl. Res.*, 42:326–338, 1989.

[GJ78] Michael R. Garey and David S. Johnson. "Strong" NP-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, July 1978.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Co., San Francisco, CA, 1979.

[GL78] Georgii V. Gens and Eugenii V. Levner. Approximate algorithms for certain universal problems in scheduling theory. *Engineering Cybernetics*, 16(6):31–36, November 1978. Translated from Russian.

[GL79a] Georgii V. Gens and Eugenii V. Levner. Computational complexity of approximation algorithms for combinatorial problems. In J. Bečvář, editor, *Mathematical Foundations of Computer Science 1979: Proceedings of the* 8^{th} *Symposium*, pages 292–300. Springer-Verlag, New York, 1979. Volume 74 of *Lecture Notes in Computer Science*. Held in Olomouc, Czechoslovakia, on 3–7 September 1979.

[GL79b] Georgii V. Gens and Eugenii V. Levner. Fast approximation algorithms for knapsack type problems. In K. Iracki, K. Malanowski, and S. Walakiewicz, editors, *Proceedings of the IX IFIP Conference on Optimization Techniques, Part 2*, pages 185–194. Springer-Verlag, New York, 1979. Volume 23 of *Lecture Notes in Control and Information Sciences*. Held in Warsaw on 4–8 September 1979.

[Gen81] Georgii V. Gens. Problems with fixed payments and efficient approximate algorithms for solving them. *Engineering Cybernetics*, 19(6):14–23, November 1981. Translated from Russian.

[GL81] Georgii V. Gens and Eugenii V. Levner. Fast approximation algorithm for job sequencing with deadlines. *Discrete Applied Math.*, 3(4):313–318, November 1981.

[Geo67] Arthur M. Geoffrion. Solving bicriterion mathematical programs. *Opns. Res.*, 15(1):39–54, January 1967.

[Geo68] Arthur M. Geoffrion. Proper efficiency and the theory of vector maximization. *J. Math. Anal. and Appl.*, 22:618–630, 1968.

[HZ80] Gabriel Y. Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–310, Winter 1980.

[Han80] Pierre Hansen. Bicriterion path problems. In Günter Fandel and Tomas Gal, editors, *Multiple Criteria Decision Making Theory and Application*, pages 109–127. Springer-Verlag, New York, 1980. Volume 177 of *Lecture Notes in Economics and Mathematical Systems.* Proceedings of the Third Conference, Hagen/Königswinter, West Germany, 20–24 August 1979.

[Har83] Roger Hartley. Survey of algorithms for vector optimisation problems. In Simon French, Roger Hartley, L. C. Thomas, and Douglas J. White, editors, *Multi-Objective Decision Making*, pages 1–34. Academic Press, New York, 1983. Proceedings of a conference on multi-objective decision making. Held at the University of Manchester on 20–22 April 1982.

[Har85a] Roger Hartley. Linear multiple objective programming. In Paolo Serafini, editor, *Mathematics of Multi Objective Optimization*, pages 157–177. Springer-Verlag, New York, 1985. CISM Courses and Lectures, Number 289, International Centre for Mechanical Sciences. Proceedings of Mathematics of Multi Objective Optimization, International Centre for Mechanical Sciences, Udine, Italy, 3–4 September 1984.

[Har85b] Roger Hartley. Vector optimal routing by dynamic programming. In Paolo Serafini, editor, *Mathematics of Multi Objective Optimization*, pages 215–224. Springer-Verlag, New York, 1985. CISM Courses and Lectures, Number 289, International Centre for Mechanical Sciences. Proceedings of Mathematics of Multi Objective Optimization, International Centre for Mechanical Sciences, Udine, Italy, 3–4 September 1984.

[Hen83] Mordechai I. Henig. Vector-valued dynamic programming. *SIAM J. Control and Optimization*, 21(3):490–499, May 1983.

[Hen85a] Mordechai I. Henig. Applicability of the functional equation in multi criteria dynamic programming. In Paolo Serafini, editor, *Mathematics of Multi Objective Optimization*, pages 189–213. Springer-Verlag, New York, 1985. CISM Courses and Lectures, Number 289, International Centre for Mechanical Sciences. Proceedings of Mathematics of Multi Objective Optimization, International Centre for Mechanical Sciences, Udine, Italy, 3–4 September 1984.

[Hen85b] Mordechai I. Henig. The principle of optimality in dynamic programming with returns in partially ordered sets. *Math. of Opns. Res.*, 10(3):462–470, August 1985.

[Hen86] Mordechai I. Henig. The shortest path problem with two objective functions. *Eur. J. Opnl. Res.*, 25(2):281–291, May 1986. Volume year is 1985.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Co., Reading, MA, 1979.

[HS74] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, April 1974.

[HS76] Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *J. ACM*, 22(2):317–327, April 1976.

[HM79] Ching-Lai Hwang and Abu Syed Md. Masud. *Multiple Objective Decision Making—Methods and Applications*, volume 164 of *Lecture Notes in Economics and Mathematical Systems.* Springer-Verlag, New York, 1979.

[IHTI78] Toshihide Ibaraki, Toshiharu Hasegawa, Katsumi Teranaka, and Jiro Iwase. The multiple-choice knapsack problem. *Journal of the Operations Research Society of Japan*, 21(1):59–93, March 1978.

[Iba80] Toshihide Ibaraki. Approximate algorithms for the multiple-choice continuous knapsack problems. *Journal of the Operations Research Society of Japan*, 23(1):28–62, March 1980.

[IK75] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, October 1975.

[IK78] Oscar H. Ibarra and Chul E. Kim. Approximation algorithms for certain scheduling problems. *Math. of Opns. Res.*, 3(3):197–204, August 1978.

[Jah85] Johannes Jahn. Scalarization in multi objective optimization. In Paolo Serafini, editor, *Mathematics of Multi Objective Optimization*, pages 45–88. Springer-Verlag, New York, 1985. CISM Courses and Lectures, Number 289, International Centre for Mechanical Sciences. Proceedings of Mathematics of Multi Objective Optimization, International Centre for Mechanical Sciences, Udine, Italy, 3–4 September 1984.

[Joh74] David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comp. and Systems Sci.*, 9(3):256–278, 1974.

[Kal86] Ignacy Kaliszewski. Generating nested subsets of efficient solutions. In J. Jahn and W. Krabs, editors, *Recent Advances and Historical Development of Vector Optimization*, pages 173–182. Springer-Verlag, New York, 1986. Volume 294 of *Lecture Notes in Economics and Mathematical Systems*. Proceedings of an International Conference on Vector Optimization. Held in Darmstadt, FRG, on 4–7 August 1986.

[KK84] Ravi Kannan and Bernhard H. Korte. Approximative combinatorial algorithms. In Richard W. Cottle, Milton L. Kelmanson, and Bernhard H. Korte, editors, *Mathematical Programming*, pages 195–248. North-Holland Publishing Co., New York, 1984. Proceedings of the International Conference on Mathematical Programming, Rio de Janeiro, 6–8 April 1981.

[KI87] Naoki Katoh and Toshihide Ibaraki. A parametric characterization and an ε-approximation scheme for the minimization of a quasiconcave program. *Discrete Applied Math.*, 17:39–66, 1987.

[Kor79] Bernhard H. Korte. Approximative algorithms for discrete optimization problems. In Peter L. Hammer, Ellis L. Johnson, and Bernhard H. Korte, editors, *Discrete Optimization I*, pages 85–120. North-Holland Publishing Co., New York, 1979. Published as volume 4 of *Annals of Discrete Mathematics*. Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications. Held in Banff and Vancouver, Canada, August 1977.

[KS81] Bernhard H. Korte and Rainer Schrader. On the existence of fast approximation schemes. In Olvi L. Mangasarian, Robert R. Meyer, and Stephen M. Robinson, editors, *Nonlinear Programming 4*, pages 415–437. Academic Press, New York, 1981. Proceedings of Nonlinear Programming Symposium 4, University of Wisconsin—Madison, 14–16 July 1980.

[Law76] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[Law79] Eugene L. Lawler. Fast approximation algorithms for knapsack problems. *Math. of Opns. Res.*, 4(4):339–356, November 1979.

[LM79] R. Loulou and E. Michaelides. New greedy-like heuristics for the multidimensional 0-1 knapsack problem. *Opns. Res.*, 27(6):1101–1114, November 1979.

[MC84] Michael J. Magazine and Maw-Sheng Chern. A note on approximation schemes for multidimensional knapsack problems. *Math. of Opns. Res.*, 9(2):244–247, May 1984.

[MM78] R. E. Marsten and T. L. Morin. A hybrid approach to discrete mathematical programming. *Math. Prog.*, 14:21–44, 1978.

[Mar84a] Ernesto Queirós Vieira Martins. On a multicriteria shortest path problem. *Eur. J. Opnl. Res.*, 16:236–245, 1984.

[Mar84b] Ernesto Queirós Vieira Martins. On a special class of bicriterion path problems. *Eur. J. Opnl. Res.*, 17:85–94, 1984.

[MP89] S. Thomas McCormick and Michael L. Pinedo. Scheduling n independent jobs on m uniform machines with both flow time and makespan objectives: A parametric analysis. Does not include best time bounds., January 1989.

[Meg79] Nimrod Megiddo. Combinatorial optimization with rational objective functions. *Math. of Opns. Res.*, 4(4):414–424, November 1979.

[ME74] T. L. Morin and A. M. O. Esogbue. The imbedded state space approach to reducing dimensionality in dynamic programs of higher dimensions. *J. Math. Anal. and Appl.*, 48:801–810, 1974.

[MM76] T. L. Morin and R. E. Marsten. An algorithm for nonlinear knapsack problems. *Management Sci.*, 22(10):1147–1158, June 1976.

[NWF78] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Math. Prog.*, 14:265–294, 1978.

[Orl82] James B. Orlin. Fast approximation schemes for combinatorial optimization problems. Presentation at TIMS/ORSA meeting, San Diego, 1982.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[PST91] Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. 1991.

[Ros85] Richard E. Rosenthal. Principles of multiobjective optimization. *Decision Sciences*, 16(2):133–152, Spring 1985.

[RV81] Bernard Roy and Philippe Vincke. Multicriteria analysis: Survey and new directions. *Eur. J. Opnl. Res.*, 8:207–218, 1981.

[Ruh88] Günther Ruhe. Complexity results for multicriterial and parametric network flows using a pathological graph of Zadeh. *Zeitschrift für Operations Research, Series A: Theory*, 32:9–27, 1988.

[RF89] Günther Ruhe and Bernd Fruhwirth. ε-optimality for bicriteria programs and its application to minimum cost flows. Report 1989-140, Institut für Mathematik, Technische Universität Graz, Austria, August 1989.

[SO94] Hershel M. Safer and James B. Orlin. Fast approximation schemes for multi-criteria flow, knapsack, and scheduling problems. October 1994.

[Sah75] Sartaj Sahni. Approximate algorithms for the 0/1 knapsack problem. *J. ACM*, 22(1):115–124, January 1975.

[Sah76] Sartaj Sahni. Algorithms for scheduling independent tasks. *J. ACM*, 22(1):116–127, January 1976.

[SG76] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, July 1976.

[Sah77] Sartaj Sahni. General techniques for combinatorial approximation. *Opns. Res.*, 25(6):920–936, November 1977.

[San86] N. G. F. Sancho. A multi-objective routing problem. *Engineering Optimization*, 10(2):71–76, 1986.

[Sch83] Rainer Schrader. Approximations to clustering and subgraph problems on trees. *Discrete Applied Math.*, 6(3):301–309, September 1983.

[Ser84] Paolo Serafini. Dual relaxation and branch-and-bound techniques for multi-objective optimization. In M. Grauer and A. P. Wierzbicki, editors, *Interactive Decision Analysis*, pages 84–90. Springer-Verlag, New York, 1984. Volume 229 of *Lecture Notes in Economics and Mathematical Systems.* Proceedings of an International Workshop on Interactive Decision Analysis and Interpretative Computer Intelligence. Held at the International Institute for Applied Systems Analysis, Laxenburg, Austria, on 20–23 September 1983.

[Ser85a] Paolo Serafini, editor. *Mathematics of Multi Objective Optimization.* CISM Courses and Lectures, Number 289, International Centre for Mechanical Sciences. Springer-Verlag, New York, 1985. Proceedings of Mathematics of Multi Objective Optimization, International Centre for Mechanical Sciences, Udine, Italy, 3–4 September 1984.

[Ser85b] Paolo Serafini. A unified approach for scalar and vector optimization. In Paolo Serafini, editor, *Mathematics of Multi Objective Optimization*, pages 89–104. Springer-Verlag, New York, 1985. CISM Courses and Lectures, Number 289, International Centre for Mechanical Sciences. Proceedings of Mathematics of Multi Objective Optimization, International Centre for Mechanical Sciences, Udine, Italy, 3–4 September 1984.

[Ser87] Paolo Serafini. Some considerations about computational complexity for multi objective combinatorial problems. In J. Jahn and W. Krabs, editors, *Recent Advances and Historical Development of Vector Optimization*, pages 222–232. Springer-Verlag, New York, 1987. Volume 294 of *Lecture Notes in Economics and Mathematical Systems.* Proceedings of an International Conference on Vector Optimization. Held in Darmstadt, FRG, on 4–7 August 1986.

[SP87] I. V. Sergienko and V. A. Perepelitsa. Finding the set of alternatives in discrete multicriterion problems. *Cybernetics*, 23(5):673–683, September 1987. Translated from Russian.

[Sha76] Jeremy F. Shapiro. Multiple criteria public investment decision making by mixed integer programming. In H. Thiriez and S. Zionts, editors, *Multiple Criteria Decision Making*, pages 170–181. Springer-Verlag, New York, 1976. Volume 130 of *Lecture Notes in Economics and Mathematical Systems.* Proceedings of a conference. Held in Jouy-en-Josas, France, on 21–23 May 1975.

[SOVM88] Bala Shetty, David L. Olson, M. A. Venkataramanan, and Ishwar Murthy. Network reoptimization procedures for multiobjective network problems. Manuscript., 1988.

[Sni88] Moshe Sniedovich. A multi-objective routing problem revisitied. *Engineering Optimization*, 13(2):99–108, 1988.

[SZ77] Martin K. Starr and Milan Zeleny, editors. *Multiple Criteria Decision Making*, volume 6 of *TIMS Studies in the Management Sciences.* North-Holland Publishing Co., New York, 1977.

[Ste86] Ralph E. Steuer. *Multiple Criteria Optimization: Theory, Computation, and Application.* John Wiley & Sons, New York, 1986.

[Tar83] Robert E. Tarjan. *Data Structures and Network Algorithms.* CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, 1983.

[Vav90] Stephen A. Vavasis. Approximation algorithms for concave quadratic programming. Technical Report 90-1172, Department of Computer Science, Cornell University, December 1990.

[VK81] Bernardo Villareal and Mark H. Karwan. Multicriteria integer programming: A (hybrid) dynamic programming recursive approach. *Math. Prog.*, 21:204–223, 1981.

[VK82] Bernardo Villareal and Mark H. Karwan. Multicriteria dynamic programming with an application to the integer case. *J. Opt. Theory and Appl.*, 38(1):43–69, September 1982.

[War83] Arthur Warburton. Exact and approximate solution of multiple objective shortest path problems. Working Paper 83-74, University of Ottawa, 1983.

[War87] Arthur Warburton. Approximation of pareto optima in multiple-objective, shortest-path problems. *Opns. Res.*, 35(1):70–79, January 1987.

[Whi69] Douglas J. White. *Dynamic Programming.* Holden-Day, San Francisco, California, 1969.

[Whi80] Douglas J. White. Generalized efficient solutions for sums of sets. *Opns. Res.*, 28(3):844–846, May 1980.

[Whi82a] Douglas J. White. *Optimality and Efficiency.* John Wiley & Sons, 1982.

[Whi82b] Douglas J. White. The set of efficient solutions for multiple objective shortest path problems. *Computers and Opns. Res.*, 9(2):101–107, 1982.

[Whi86] Douglas J. White. Epsilon efficiency. *J. Opt. Theory and Appl.*, 49(2):319–337, May 1986.

[YS81] P. L. Yu and L. Seiford. Multistage decision problems with multiple criteria. In Peter Nijkamp and Jaap Spronk, editors, *Multiple Criteria Analysis: Operational Methods*, chapter 14, pages 235–244. Gower Publishing Company, Limited, Aldershot, England, 1981.

[Zem81] Eitan Zemel. Measuring the quality of approximate solutions to zero-one programming problems. *Math. of Opns. Res.*, 6(3):319–332, August 1981.

[Zio78] Stanley Zionts, editor. *Multiple Criteria Problem Solving*, volume 155 of *Lecture Notes in Economics and Mathematical Systems.* Springer-Verlag, New York, 1978. Proceedings of a conference in Buffalo, NY, on 22–26 August 1977.

[Zio79] Stanley Zionts. A survey of multiple criteria integer programming methods. In Peter L. Hammer, Ellis L. Johnson, and Bernhard H. Korte, editors, *Discrete Optimization II*, pages 389–398. North-Holland Publishing Co., New York, 1979. Volume 5 of *Annals of Discrete Mathematics.* Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications, Held in Banff and Vancouver, Canada, August 1977.