Fast Approximation Schemes for Multi-Criteria Flow, Knapsack, and Scheduling Problems

by
Hershel M. Safer
James B. Orlin

WP #3757-95 January 1995

Fast Approximation Schemes For Multi-Criteria Flow, Knapsack, and Scheduling Problems

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 Flow, Knapsack, and Scheduling Problems

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. A solution in this case typically consists of multiple routes because, in general, no point is both shortest and cheapest. Finding the efficient frontier or tradeoff curve for such a multi-criteria problem is often difficult, in part because the solution set can have exponentially many points. As a result, multi-criteria problems are often solved by *approximating* the efficient frontier.

A fast approximation scheme (FAS) for a multi-criteria problem is an algorithm that can quickly find an arbitrarily-good approximation to the problem's efficient frontier. Necessary and sufficient conditions for the existence of an FAS for a problem were introduced in [SO94]. The conditions are stated in terms of the existence of a V-pseudo-polynomial (VPP) time algorithm for the problem. A new form of reducibility is also introduced and shown to be useful for studying the existence of FAS's.

This paper extends the work of [SO94] by applying it to a variety of discrete optimization problems. The Source-to-Tree (STT) Network Flow problem is introduced. This problem is interesting because it generalizes many commonly-treated combinatorial optimization problems. A VPP time algorithm is demonstrated for a particular STT problem and is used to show that the problem has an FAS.

Results are also derived about the computational complexity of finding approximate solutions to several multi-criteria knapsack, scheduling, and production planning problems. Many theorems extend previously-known results to multi-criteria problems. For some problems, results about problems with binary variables are extended to problems with general integer variables. These and other results are of interest even for problems with only a single criterion.

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)

$M_{\text{V}}(I)$ The largest value of instance I. ([1], section 3.2)

$\overline{M}_{\text{V}}(I)$ A close upper bound on $M_{\text{V}}(I)$ 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 $f : S \rightarrow \mathcal{Z}^{r+}$, 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 $O\left(\left(L(I)/\varepsilon\right)^k\right)$, 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 $M_{\mathrm{V}}(I)$. ([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 $f : S \rightarrow \mathcal{Z}^{r+}$. ([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. If, however, each arc has *both* a distance *and* a cost, then one might ask for routes that are *simultaneously* short and inexpensive. A solution in this case typically consists of multiple routes because, in general, no point is optimal for both criteria. Finding the efficient frontier for such a multi-criteria problem is often difficult, in part because it can have exponentially many points. As a result, such problems are often solved by *approximating* the efficient frontier.

Necessary and sufficient conditions for the existence of a fast approximation scheme (FAS) for such a problem were introduced in [SO94]. The conditions are stated in terms of the existence of a V-pseudo-polynomial (VPP) time algorithm for the problem. In addition, a new form of reducibility is introduced and shown to be useful for studying the existence of FAS's. The present paper applies the results of [SO94] to analyze the complexity of solving a variety of combinatorial optimization problems.

The Source-to-Tree (STT) Network Flow problem is introduced in Section 2. To the knowledge of the authors this problem has not been considered elsewhere. This problem is interesting, however, because later in this paper it is shown to generalize a wide variety of commonly treated discrete optimization problems. Several useful classes of objective functions are defined in Section 3, and a VPP algorithm is demonstrated for a class of STT Network Flow problems in Section 4. The existence of this algorithm is used to show that the problem also has an FAS. The boundary between easy and difficult classes of STT Network Flow problems is explored in Section 5.

The STT Network Flow problem is used to assess the complexity of finding approximate solutions to several knapsack, scheduling, and production planning problems in Sections 6 through 9. Most problems discussed in those sections have been previously considered with only a single criterion; this paper extends the previously-known results to multi-criteria versions of the problems. In addition, the algorithms described here accommodate the general integer case of some problems for which only the 0-1 case has been considered until now. FAS's are also demonstrated for several problems that have not been considered elsewhere.

The existence of an FAS for a general lot-sizing problem is demonstrated in Section 9. Although pseudo-polynomial time algorithms were previously known for this problem, they cannot be easily transformed into FAS's. The results of Section 9 indicate that this problem has a pseudo-polynomial time algorithm that may be more useful in some situations than are previously-known algorithms.

2 The Source-To-Tree Network Flow Feasible Set

The Source-to-Tree Network Flow problem is introduced in this section. Not only is this problem interesting for its own sake, but it is also useful for proving the existence of an FAS for a variety of other problems. As shown in Sections 6 through 9, it generalizes quite a few knapsack, job scheduling, and lot-sizing problems.

Because it unifies a large number of interesting problems, results for the Source-to-Tree Network Flow problem have wide applicability.

This section explains how to specify feasible sets for particular classes of Source-To-Tree Network Flow problems. Objective functions are discussed in the next section. A VPP algorithm for one class of problems is presented in Section 4.1; results for other classes are obtained in Section 5.

To start, recall two definitions. A *generalized network* is a network in which the flow that enters at the tail of an arc is multiplied by an arc-specific constant before it emerges at the head of the arc[AMO93]. Two directed arcs are *anti-parallel* if they connect the same pair of nodes but are oriented in opposite directions.

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

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

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

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

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

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

sufficient	if $e_v(x) \geq 0$,
deficient	if $e_v(x) \leq 0$, or
exact	if $e_v(x) = 0$.

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

Many classes of STT networks are interesting. Problems will defined on some particular classes in Sections 4.1 and 5. A class can be identified using a 5-tuple in which each component specifies one of the following details about the networks in the class.

1. Restrictions on parallel arcs: Parallel arcs may or may not be allowed in the tree part of the network. This restriction refers to the directed arcs in G, not to the undirected edges mentioned in the definition of an STT network. Excluding parallel arcs does not preclude the use of anti-parallel arcs. This restriction does not apply to arcs to or from node 0.

2. Restrictions on arc flow capacities: The arcs of a network can have either unit capacities or general non-negative integer capacities.

3. Restrictions on arc flow multipliers: Multipliers can either be unity, as in an ordinary network, or general non-negative integers, as in a generalized network.

4. Restrictions on node demands: A network may consist of all transshipment nodes, all demand nodes, all supply nodes, or a mixture. These descriptions do not apply to node 0.

5. Restrictions on node excesses for feasible flows: A flow may be required to be sufficient, deficient, or exact. Note that this is a condition on acceptable solutions, whereas the first four conditions refer to characteristics of the graph structure.

The possible values for each component of the 5-tuple are listed in Figure 1. For example, the class of networks $\mathcal{C} = (A1/CG/MG/NM/FS)$ consists of STT networks with no directed parallel arcs, capacities and multipliers that can be any non-negative integers, both supply and demand nodes, and in which the flow to each node must be sufficient. This class of STT networks is discussed further in Section 4.1.

A flow $x \in \mathcal{Z}^{|A|+}$ is *feasible* if it satisfies the flow requirements (the fifth component of the STT class definition) and if, in addition, $x \leq u$. The set of feasible flows for an instance constitutes the feasible set S for that instance; the integer program that specifies the set is generally not written out explicitly. The collection of feasible sets S for the networks in a class of STT networks is the family of feasible sets Ψ for an STT Network Flow problem defined on the class.

This section has described feasible sets for STT Network Flow problems. In order to specify an STT Network Flow optimization or feasibility problem completely, other parts of the problem must be specified as well: a family $\mathcal{F}$ of r-criteria objective functions defined on the vector x of flows, an r-dimensional direction vector ω, and, for a feasibility problem, an r-dimensional target vector. The next section discusses some appropriate classes of objective functions, and Section 4.1 presents a VPP algorithm for a problem defined on a particular class of STT networks.

Arcs:	$A1$	No parallel arcs are allowed in the tree part of the network.
	AP	Parallel arcs are allowed.
Capacities:	$C1$	Unit flow capacities ($u = 1$).
	CG	General flow capacities in $\mathcal{Z}^+$.
Multipliers:	$M1$	Unit flow multipliers ($\mu = 1$).
	MG	General flow multipliers in $\mathcal{Z}^+$.
Nodes:	$N0$	Network may contain only transshipment nodes ($b = 0$).
	ND	Network may contain only demand nodes ($b \geq 0$).
	NS	Network may contain only supply nodes ($b \leq 0$).
	NM	Network may contain a mixed node set (no restriction on b).
Flows:	FS	Flow into each node must be sufficient.
	FD	Flow into each node must be deficient.
	FE	Flow into each node must be exact.
	FU	Flow excesses are unrestricted.

Figure 1: Components of STT network class specification.

3 Objective Functions

The problems discussed in the remainder of this paper are distinguished from each other primarily by the structures of their feasible regions. Before looking at these feasible regions, however, some attention should be paid to the objective functions that are used. This section examines several kinds of objective functions that arise frequently in practice.

This section does not deal with an important aspect of objective functions: whether they are to be minimized or maximized. The difference is crucial when considering approximation algorithms. Consider the Traveling Salesman problem on complete graphs without the triangle inequality. The minimum cost version has no polynomial time ε-approximate algorithm for any $\varepsilon > 0$ unless $\mathcal{P} = \mathcal{NP}$[SG76]. A tour of length at least one-half the optimum can easily be found in the maximization case, however, by extending a maximum weight matching.

Another example is the Multi-Dimensional Binary Knapsack problem discussed in [SO94]. As mentioned there, the maximization form has no FAS unless $\mathcal{P} = \mathcal{NP}$. The minimization form, however, can be solved exactly and quickly by setting $x = 0$, that is, leaving the knapsack empty.

Section 3.1 generalizes the notion of monotonically increasing functions. This generalization is applied to additively separable functions in Section 3.2. VPP reductions between problems with difference function classes are derived in Section 3.3.

3.1 Monotonicity

Many common objective functions are monotonically increasing or monotonically decreasing. A function $f : \mathcal{Z}^{n+} \rightarrow \mathcal{Z}^{r+}$ is *monotonically increasing* if for $x, y \in \mathcal{Z}^{n+}$, $x \leq y$ implies that $f(x) \leq f(y)$. The " $\leq$ "

relation is applied on a component-by-component basis. The function f is *monotonically decreasing* if the function $g = -f$ is monotonically increasing.

Results for these kinds of functions have been derived in the single-criterion case. Restricting attention to, say, monotonically increasing objective functions in the multi-criteria case, however, is not as useful, because different components of an objective function can be optimized in different directions. One way to generalize single-criterion monotonic functions is to require that the values of all components of the objective function simultaneously improve or get worse when the components of the argument increase, as illustrated by the following definitions.

Definition 1 An objective function $f : \mathcal{Z}^{n+} \rightarrow \mathcal{Z}^{r+}$ is *order-preserving* w.r.t. a direction vector $\omega \in \Omega^r$ if either of the following two conditions holds:

$$\text{(a)} \quad \forall x, y \in \mathcal{Z}^{n+},\ x \leq y \Longrightarrow \text{ for } i = 1, \ldots, r, \quad \begin{cases} \omega_i = \text{“} \leq \text{”} & \Rightarrow \quad f_i(x) \leq f_i(y)\ , \\ \omega_i = \text{“} \geq \text{”} & \Rightarrow \quad f_i(x) \geq f_i(y)\ , \\ \omega_i = \text{“} = \text{”} & \Rightarrow \quad f_i(x) \leq f_i(y)\ . \end{cases}$$

$$\text{(b)} \quad \forall x, y \in \mathcal{Z}^{n+},\ x \leq y \Longrightarrow \text{ for } i = 1, \ldots, r, \quad \begin{cases} \omega_i = \text{“} \leq \text{”} & \Rightarrow \quad f_i(x) \leq f_i(y)\ , \\ \omega_i = \text{“} \geq \text{”} & \Rightarrow \quad f_i(x) \geq f_i(y)\ , \\ \omega_i = \text{“} = \text{”} & \Rightarrow \quad f_i(x) \geq f_i(y)\ . \end{cases}$$

■

The point is that the value of each component of an order-preserving function changes in accordance with the corresponding component of the direction vector. Component functions corresponding to direction vector entries of " = " can behave as either " $\leq$ " or " $\geq$ ", but must be consistent in this regard over the entire feasible set. Furthermore, all component functions that correspond to " = " entries must behave the same way.

The definitions of "monotonically increasing" and "order preserving" could have been stated for functions that map from a subset $S \subseteq \mathcal{Z}^{n+}$ rather than from the entire space $\mathcal{Z}^{n+}$. Doing so would, however, have required a large overhead in notation to accommodate the function classes defined later in this section. The more complicated formulation would not add to the insight provided by the results in this paper, nor would it extend the applicability of the results in practice. Because of these considerations, the present definitions are used instead of more general definitions.

Definition 2 An objective function f is *order-reversing* w.r.t. a direction vector ω if the function $g = -f$ is order-preserving w.r.t. ω, and f is *order-monotone* w.r.t. a direction vector ω if f is either order-preserving or order-reversing w.r.t. ω. ■

Definition 3 The following classes of objective functions are defined for a direction vector ω:

$$\begin{aligned} \mathcal{P}(\omega) &= \{ f : f \text{ is order-preserving w.r.t. } \omega \} , \\ \mathcal{R}(\omega) &= \{ f : f \text{ is order-reversing w.r.t. } \omega \} , \\ \mathcal{M}(\omega) &= \mathcal{P}(\omega) \cup \mathcal{R}(\omega) . \end{aligned}$$

■

The notions of closure under scaling and under box constraints are introduced in [SO94]. For any direction vector ω, each of the sets $\mathcal{P}(\omega)$, $\mathcal{R}(\omega)$, and $\mathcal{M}(\omega)$ is closed under scaling and under box constraints. The following lemma describes the relationship between the sets $\mathcal{P}(\omega)$ and $\mathcal{R}(\omega)$.

Lemma 1 *Let S be an n-dimensional feasible set containing a "largest" element u that satisfies the condition $x \in S \Rightarrow (u - x) \in S$. Let $f : S \rightarrow \mathcal{Z}^{r+}$ be a function and define the function $f' : S \rightarrow \mathcal{Z}^{r+}$ by $f'(x) = f(u - x)$. Then $f \in \mathcal{P}(\omega)$ if and only if $f' \in \mathcal{R}(\omega)$.*

Furthermore, a component function f_i that corresponds to $\omega_i =$ " $=$ " behaves as though $\omega_i =$ " $\leq$ " (i.e., according to part (a) in the definition of order-preserving) if and only if the function f_i' behaves as though $\omega_i =$ " $\geq$ " (i.e., according to part (b) in the definition of order-preserving).

Proof. Suppose that $f \in \mathcal{P}(\omega)$. Let $x, y \in S$ be given, with $x \leq y$; then $(u - x), (u - y) \in S$, and $(u - x) \geq (u - y)$. For each $i = 1, \ldots, r$, consider the possible values of ω_i. If $\omega_i =$ " $\leq$ ", then

$$f_i'(x) = f_i(u - x) \geq f_i(u - y) = f_i'(y) .$$

Analogous statements are true for $\omega_i \in \{$ " $\geq$ ", " $=$ "$\}$. Therefore $f' \in \mathcal{R}(\omega)$.

The proof of the converse statement is similar. ■

3.2 Separability

The necessary and sufficient conditions in [SO94] for the existence of an FAS apply to problems with quite general objective functions. Often, though, VPP algorithms use dynamic programming, which limits the kinds of objective functions that can be accommodated easily. In order to be amenable to the application of dynamic programming, a problem must typically have a separable objective function. The most common separable objective functions are additively separable, that is, they can be written as $f(x) = \sum_{j=1}^{n} f_j(x_j)$. Functions of the form $f(x) = \prod_{j=1}^{n} f_j(x_j)$ and $f(x) = \min\{f_j(x_j), j = 1, \ldots, n\}$ are also used[Whi69, section 3.2]. This section introduces additively separable functions that are order-monotone.

The reason that separable objective functions are common in dynamic programming applications is that the value of the function can be determined incrementally by optimizing the value of one variable at a time. Although the focus here is additively separable functions, fairly general definitions of separability are

presented in [YS81, VK82, Har83]. The problems that are amenable to solution by dynamic programming are characterized in [Bel61], as cited in [LH90], as those which can be formulated as Markov processes. Note that the conditions of monotonicity and nonbounded dominance in [YS81] are satisfied by the objective functions discussed in [SO94].

Some attempts have been made to extend dynamic programming to problems with objective functions that are not separable. One approach is to consider history-remembering policies[Kre77, Hen85]. Another approach is to approximate the non-separable functions with separable functions[Sni87]. These approaches are not pursued in this work.

Definition 4 A function $f : S \to \mathcal{Z}^{r+}$, where $S \subseteq \mathcal{Z}^{n+}$, is *additively separable* if there exist functions f_1, ..., f_n such that for all $x \in S$, $f(x) \equiv \sum_{j=1}^{n} f_j(x_j)$. ■

The discussion in Section 3.1 can be specialized to this kind of function.

Definition 5 For a direction vector $\omega \in \Omega^r$,

$$\begin{aligned} \mathcal{AP}(\omega) &= \{ f \in \mathcal{P}(\omega) : f \text{ is additively separable} \}, \\ \mathcal{AR}(\omega) &= \{ f \in \mathcal{R}(\omega) : f \text{ is additively separable} \}, \\ \mathcal{AM}(\omega) &= \mathcal{AP}(\omega) \cup \mathcal{AR}(\omega). \end{aligned}$$

■

The notions of quasi-closure under scaling and under box constraints, as well as of scaling neighbor and box constraint neighbor, were introduced in [SO94] along with the notions of closure mentioned in Section 3.1. None of the sets in Definition 5 is closed under scaling or under box constraints, but for any direction vector ω, each is quasi-closed under scaling and under box constraints.

Scaling and box constraint neighbors of a function $f \in \mathcal{AM}(\omega)$ are easy to find. Write f as $f = \sum_{j=1}^{n} f_j(x_j)$, where f_j is an r-component function. For each $j = 1, \ldots, n$ and $i = 1, \ldots, r$, let $f_{i,j}$ be the i^{th} component function of f_j. Then one scaling neighbor of f w.r.t. v is the function g with r component functions $g_i = \sum_{j=1}^{n} \lfloor f_{i,j}(x_j)/v_i \rfloor$, $i = 1, \ldots, r$. One box constraint neighbor of f w.r.t. M is the function h with r component functions $h_i = \sum_{j=1}^{n} \min\{f_{i,j}(x_j), M_i\}$, $i = 1, \ldots, r$.

The functions that constitute the terms of an additively separable function behave similarly to the entire function with regard to order monotonicity, as shown by the following lemma.

Lemma 2 *Let $S \subseteq \mathcal{Z}^{n+}$ be closed under the "$\leq$" relation. That is, for any $x, y \in \mathcal{Z}^{n+}$, with $x \leq y$, if $y \in S$ then $x \in S$. Suppose that a function $f : S \to \mathcal{Z}^{r+}$ is defined by $f(x) \equiv \sum_{j=1}^{n} f_j(x_j)$. Then*

(a) $f \in \mathcal{AP}(\omega)$ if and only if for $j = 1, \ldots, n$, $f_j \in \mathcal{P}(\omega)$, with the same behavior on $\omega_i =$ "$=$" as f, and

(b) $f \in \mathcal{AR}(\omega)$ if and only if for $j = 1, \ldots, n$, $f_j \in \mathcal{R}(\omega)$, with the same behavior on $\omega_i =$ "$=$" as f.

Proof. Only the proof for (a) is given; the proof for (b) is similar.

Necessity: Suppose that $f \in \mathcal{AP}(\omega)$. Choose $x, y \in S$ with $x \leq y$, and focus on some criterion i. Suppose that $\omega_i = \text{“} \leq \text{”}$; the proof is similar for $\omega_i \in \{\text{“} \geq \text{”}, \text{“} = \text{”}\}$.

Since $f \in \mathcal{AP}(\omega)$, $f_i(x) \leq f_i(y)$. Suppose that f_j does not behave as claimed for some $j \in \{1, \ldots, n\}$. Let $\overline{x} = (0, \ldots, 0, x_j, 0, \ldots, 0)$ and $\overline{y} = (0, \ldots, 0, y_j, 0, \ldots, 0)$, where both $\overline{x}$ and $\overline{y}$ are n-vectors with the non-zero elements in component j.

By assumption, $\overline{x}, \overline{y} \in S$. But $f(\overline{x}) = f_j(\overline{x})$ and $f(\overline{y}) = f_j(\overline{y})$, contradicting the assumption that f_j does not behave the in the same way as f.

Sufficiency: Suppose that for $j = 1, \ldots, n$, $f_j \in \mathcal{P}(\omega)$, all with the same behavior on $\omega_i = \text{“} = \text{”}$, and that $x, y \in S$ with $x \leq y$. For any $i = 1, \ldots, r$, suppose that $\omega_i = \text{“} \leq \text{”}$. Then $f_i(x) = \sum_{j=1}^{n} f_{i,j}(x_j) \leq \sum_{j=1}^{n} f_{i,j}(y_j) = f_i(y)$. Analogous statements are true for $\omega_i \in \{\text{“} \geq \text{”}, \text{“} = \text{”}\}$. So $f \in \mathcal{AP}(\omega)$, with the same behavior on $\omega_i = \text{“} = \text{”}$ as the functions f_j. ∎

3.3 Reductions for Classes of Objective Functions

Reductions can be used to show relationships between classes of objective functions. The objective function classes defined in Section 3 are of particular interest in the present work.

Consider first the classes of order-preserving and order-reversing objective functions. Problems defined with these function classes are similar in some respects. This is particularly true when the direction vectors are in $\Omega^{=}$, that is, the set of direction vectors $\{\omega^{r,=} : r = 1, \ldots\}$. The relationship between $\mathcal{P}(\omega^{r,=})$, $\mathcal{R}(\omega^{r,=})$, and $\mathcal{M}(\omega^{r,=})$, for any r, is described in the following theorem.

Theorem 1 *For any family Ψ of feasible sets, the following problems are VPP equivalent for any k:*

1. $(\Psi, \mathcal{M}(\omega^{r,=}), \omega^{r,=}, \mathit{Feas})$, $r = 1, 2, \ldots, k$.
2. $(\Psi, \mathcal{P}(\omega^{r,=}), \omega^{r,=}, \mathit{Feas})$, $r = 1, 2, \ldots, k$.
3. $(\Psi, \mathcal{R}(\omega^{r,=}), \omega^{r,=}, \mathit{Feas})$, $r = 1, 2, \ldots, k$.

Proof. For any r, both $(\Psi, \mathcal{P}(\omega^{r,=}), \omega^{r,=}, \text{Feas})$ and $(\Psi, \mathcal{R}(\omega^{r,=}), \omega^{r,=}, \text{Feas})$ are the same as, and so are VPP equivalent to, $(\Psi, \mathcal{M}(\omega^{r,=}), \omega^{r,=}, \text{Feas})$.

Assume for the moment that r-criteria problems can be VPP reduced to single-criterion problems, i.e., that $(\Psi, \mathcal{M}(\omega^{r,=}), \omega^{r,=}, \text{Feas}) \propto_{\text{VPP}} (\Psi, \mathcal{M}(\omega^{1,=}), \omega^{1,=}, \text{Feas})$; this assertion will be proved below. The problem $(\Psi, \mathcal{M}(\omega^{1,=}), \omega^{1,=}, \text{Feas})$ can be VPP reduced to each of the other listed problems as follows:

1. An r-criteria function can be considered as an $(r+1)$-criteria function with an $(r+1)^{st}$ criterion that is identically zero, so $\mathcal{M}(\omega^{r,=}) \subseteq \mathcal{M}(\omega^{r+1,=})$. Use Lemma 1 in [SO94] to conclude that $(\Psi, \mathcal{M}(\omega^{1,=}), \omega^{1,=}, \text{Feas}) \propto_{\text{VPP}} (\Psi, \mathcal{M}(\omega^{r,=}), \omega^{r,=}, \text{Feas})$.

2. As noted above, $\left(\Psi, \mathcal{M}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Feas}\right) \propto_{\text{VPP}} \left(\Psi, \mathcal{P}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Feas}\right)$. The further VPP reduction to $\left(\Psi, \mathcal{P}\left(\omega^{r,=}\right), \omega^{r,=}, \text{Feas}\right)$ follows as in (1).

3. The reduction to $\left(\Psi, \mathcal{R}\left(\omega^{r,=}\right), \omega^{r,=}, \text{Feas}\right)$ is similar to that in (2).

The theorem is proved by establishing the earlier assertion:

$\left(\Psi, \mathcal{M}\left(\omega^{r,=}\right), \omega^{r,=}, \text{Feas}\right) \propto_{\text{VPP}} \left(\Psi, \mathcal{M}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Feas}\right)$:

Let $I = \left(S, f, \omega^{r,=}, M\right) \in \left(\Psi, \mathcal{M}\left(\omega^{r,=}\right), \omega^{r,=}, \text{Feas}\right)$ be given. Define the function $h : S \rightarrow \mathcal{Z}^{+}$ by $h(x) = \sum_{i=1}^{r} f_i(x) \left[\overline{M}_{\text{V}}(I) + 1\right]^i$. The function h can be computed in VPP time. For any $v \in \mathcal{Z}^{r+}$, $f(x) = v$ if and only if $h(x) = \sum_{i=1}^{r} v_i \left[\overline{M}_{\text{V}}(I) + 1\right]^i$. Furthermore, since each direction component $\omega_i^{r,=}$ has the value " = ", either each component function f_i is monotonically increasing or each is monotonically decreasing; hence so is h. This means that $h \in \mathcal{M}\left(\omega^{1,=}\right)$. Therefore the instance $\left(S, h, \omega^{1,=}, \sum_{i=1}^{r} M_i \left[\overline{M}_{\text{V}}(I) + 1\right]^i\right) \in \left(\Psi, \mathcal{M}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Feas}\right)$. ■

Other relationships become apparent when the directions of optimization are reversed.

Definition 6 The *dual* ω^D of a direction vector ω is obtained by changing " $\leq$ " entries in ω to " $\geq$ " and vice versa. ■

Example 1 (Dual direction vector) Here is an example of taking the dual of a direction vector:

$$\begin{pmatrix} \leq \\ \geq \\ = \end{pmatrix}^{\text{D}} = \begin{pmatrix} \geq \\ \leq \\ = \end{pmatrix} .$$

Also note that for any r, $\left(\omega^{r,=}\right)^D = \omega^{r,=}$. ■

Lemma 3 *For any family* Ψ *of feasible sets and direction vector* ω,

$$\begin{aligned} &(a) \quad \left(\Psi, \mathcal{P}(\omega), \omega, \mathit{Feas}\right) \equiv_{\text{VPP}} \left(\Psi, \mathcal{R}(\omega), \omega^D, \mathit{Feas}\right) . \\ &(b) \quad \left(\Psi, \mathcal{P}(\omega), \omega^D, \mathit{Feas}\right) \equiv_{\text{VPP}} \left(\Psi, \mathcal{R}(\omega), \omega, \mathit{Feas}\right) . \end{aligned}$$

Proof. Only the "only if" part of (a) is proved; the "if" part of (a) and both parts of (b) can be proved analogously. Let $\left(S, f, \omega, M\right) \in \left(\Psi, \mathcal{P}(\omega), \omega, \text{Feas}\right)$ be given. Let $f' = \min\{f,\ M + e_r\}$. Since $\mathcal{P}(\omega)$ is closed under box constraints, $f' \in \mathcal{P}(\omega)$, and $D_\omega\left(f'(x), M\right)$ if and only if $D_\omega\left(f(x), M\right)$.

Define $g = M + e_r - f'$; then $g \in \mathcal{R}(\omega)$. For any $x \in S$, $D_\omega\left(f'(x), M\right)$ if and only if $D_{\omega^D}\left(g(x), e_r\right)$. So solving the instance $\left(S, g, \omega^D, e_r\right) \in \left(\Psi, \mathcal{R}(\omega), \omega^D, \text{Feas}\right)$ yields a solution to the instance $\left(S, f, \omega, M\right) \in \left(\Psi, \mathcal{P}(\omega), \omega, \text{Feas}\right)$. ■

Similar relationships are true when consideration is restricted to additively separable functions. The proofs of the following theorem and lemma are analogous to the proofs of Theorem 1 and Lemma 3.

Theorem 2 *For any family Ψ of feasible sets, all the following problems are VPP equivalent:*

1. $(\Psi, \mathcal{AM}(\omega^{r,=}), \omega^{r,=}, Feas)$, $r = 1, 2, \ldots$.
2. $(\Psi, \mathcal{AP}(\omega^{r,=}), \omega^{r,=}, Feas)$, $r = 1, 2, \ldots$.
3. $(\Psi, \mathcal{AR}(\omega^{r,=}), \omega^{r,=}, Feas)$, $r = 1, 2, \ldots$.

Lemma 4 *For any family Ψ of feasible sets and direction vector $\omega \in \Omega^r$,*

$$(a) \quad (\Psi, \mathcal{AP}(\omega), \omega, Feas) \equiv_{\text{VPP}} (\Psi, \mathcal{AR}(\omega), \omega^D, Feas) .$$
$$(b) \quad (\Psi, \mathcal{AP}(\omega), \omega^D, Feas) \equiv_{\text{VPP}} (\Psi, \mathcal{AR}(\omega), \omega, Feas) .$$

4 Some Simple STT Network Flow Problems

The class of STT networks $\mathcal{C} = (A1/CG/MG/NM/FS)$ is described in Section 2. Section 4.1 focuses on a canonical subset $\mathcal{C}'$ of that class. A VPP algorithm is presented for a feasibility problem defined on $\mathcal{C}'$ and is used to prove the existence of an FAS for the optimization form of that problem. VPP reduction is used to explore the relationship between $\mathcal{C}$ and $\mathcal{C}'$ in Section 4.2.

4.1 A VPP Algorithm

A network in $\mathcal{C}$ is in canonical form if a single arc connects node 0 to each other node, each other connected pair of nodes is connected by a single pair of anti-parallel arcs, and the tree part of the network is connected. More formally, a network $G \in \mathcal{C}$ is in *canonical form* if it satisfies the following two conditions:

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

The collection of networks of $\mathcal{C}$ in canonical form is denoted $\mathcal{C}'$.

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

In a *postorder* traversal of a tree, each subtree of the root is traversed (recursively, in postorder) and then the root is visited[Knu73, RND77]. Choose an arbitrary node ρ as the root of T. Label the nodes according to any postorder ordering w.r.t. the root ρ, representing the label of node v by $l(v)$. In particular, $l(\rho) = n$. The VPP dynamic programming algorithm to be described processes the nodes of $N - \{0\}$ in increasing label order. A good overview of this approach can be found in [JN83], from which some notation is adapted.

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

A VPP algorithm that solves the feasibility problem $\left(\mathcal{C}', \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Feas}\right)$ will be presented. For purposes of stating the algorithm, say that a flow x in G is *feasible* w.r.t. $G[v, j]$ if it satisfies the following three conditions:

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

Define $\theta(v, j, k)$ to be the maximum excess flow into node v over all flows x that both are feasible w.r.t. $G[v, j]$ and have $f(x) = k$. The algorithm attacks the flow problem indirectly in that the computational focus is the maximum node excesses θ rather than the objective function f. This approach works because of the following lemma.

The idea behind this lemma is to examine $\theta(\rho, d_\rho, M)$. The flows considered in determining the value of $\theta(\rho, d_\rho, M)$ are, by the definition of θ, feasible on the subtrees of the root node ρ. Such a flow is feasible for the *entire* graph G if and only if the excess flow into ρ is sufficient, that is, if and only if $\theta(\rho, d_\rho, M) \geq 0$.

Lemma 5 *An instance* $(G, f, \text{“}=\text{”}, M)$ *of the problem* $\left(\mathcal{C}', \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Feas}\right)$ *has a solution if and only if* $\theta(\rho, d_\rho, M) \geq 0$.

Proof.

Sufficiency: Suppose that the instance $(G, f, \text{“}=\text{”}, M)$ has a solution x. Then x is a feasible flow, so

1. The flow into each node of $G[\rho, d_\rho]$ is sufficient (*including* the flow into ρ).
2. All arcs are in $G[\rho, d_\rho]$.
3. $x \in \mathcal{Z}^{|A|+}$ and $x \leq u$.

So x is feasible w.r.t. $G[\rho, d_\rho]$ and $f(x) = M$. Furthermore, since the flow into ρ is sufficient, $e_\rho(x) \geq 0$. Since $\theta(\rho, d_\rho, M)$ is the maximum of a set that contains $e_\rho(x)$, it must be that $\theta(\rho, d_\rho, M) \geq 0$.

Necessity: Suppose that $\theta(\rho, d_\rho, M) \geq 0$. This implies the existence of a flow x that is feasible w.r.t. $G[\rho, d_\rho] = G$ and satisfies $f(x) = M$. Furthermore, $e_\rho(x) = \theta(\rho, d_\rho, M) \geq 0$, so the flow into *every* node is

```
    for i = 1, ..., n
        Let v be the node with l(v) = i.
        for j = 0, ..., d_v
            for k = 0, ..., M
(*)                 Compute θ(v, j, k).
    if θ(ρ, d_ρ, M) ≥ 0
        then x is the desired solution.
        else the problem is infeasible.
```

Figure 2: The algorithm for $\left(\mathcal{C}', \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Feas}\right)$.

sufficient. Therefore x is a feasible flow that solves the instance $(G, f, \text{“}=\text{”}, M)$. ■

The algorithm for solving $\left(\mathcal{C}', \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Feas}\right)$ appears in Figure 2. A detailed description of the main computation step, labeled $(*)$, appears next. This is followed by a proof that the algorithm is VPP for $\left(\mathcal{C}', \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Feas}\right)$.

The computation for $(*)$ has two cases, depending on whether $j = 0$ or $j > 0$. When $j = 0$, perform the computation by solving the following program.

$$\begin{aligned} \theta(v,0,k) = \max_{x_{0,v}} \quad & \mu_{0,v} x_{0,v} - b_v \\ \text{s.t.} \quad & f_{0,v}\left(x_{0,v}\right) = k \\ & x_{0,v} \leq u_{0,v} \\ & x_{0,v} \in \mathcal{Z}^+ \end{aligned} \tag{1}$$

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

$$\theta(v,j,k) = \max_{\alpha,\, \beta,\, \gamma,\, \delta,\, x_{v,w},\, x_{w,v}} \quad \theta(v, j-1, \alpha) + \mu_{w,v} x_{w,v} - x_{v,w} \tag{2a}$$

$$\text{s.t.} \quad \theta(w, d_w, \beta) + \mu_{v,w} x_{v,w} - x_{w,v} \geq b_w \tag{2b}$$

$$f_{v,w}\left(x_{v,w}\right) = \gamma \tag{2c}$$

$$f_{w,v}\left(x_{w,v}\right) = \delta \tag{2d}$$

$$\alpha + \beta + \gamma + \delta = k \tag{2e}$$

$$x_{v,w} \leq u_{v,w} \tag{2f}$$

$$x_{w,v} \leq u_{w,v} \tag{2g}$$

$$\alpha,\ \beta,\ \gamma,\ \delta,\ x_{v,w},\ x_{w,v} \in \mathcal{Z}^+ \tag{2h}$$

Theorem 3 *The algorithm in Figure 2 is VPP for the feasibility problem* $\left(\mathcal{C}', \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Feas}\right)$.

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

Correctness: In the case that $j = 0$, the maximum excess on $G[v, 0]$ must be computed. This is just the subgraph of G induced by the nodes 0 and v, which, because $G \in \mathcal{C}'$, contains a single arc, $(0, v)$. The

quantity $\mu_{0,v}x_{0,v} - b_v$ is the excess flow into node v on $G[v,0]$ when the flow on arc $(0,v)$ is $x_{0,v}$. The program specified by Equations 1 maximizes this excess subject to the feasibility conditions, and so correctly computes $\theta(v,0,k)$.

In the case that $j > 0$, the maximum excess on $G[v,j]$ must be computed. Note that the nodes are considered in increasing label order, the postorder label of a node is greater than the labels of its children, and the children of a node are processed in increasing label order; therefore the values of $\theta(\cdot,\cdot,\cdot)$ that appear in Equations 2a and 2b are computed before they are used.

The structure of Equations 2 comes from decomposing the flow in $G[v,j]$ and the value k into four parts: the flow in $G[v,j-1]$, with value α; the flow in $G[w,d_w]$, with value β; and the flows on arcs (v,w) and (w,v), with values γ and δ. The total required value of k can be partitioned in this way because additively separable objective functions are used.

The objective function, Equation 2a, expresses the value of the excess flow into node v, given the flows on arcs (v,w) and (w,v) and in the first $(j-1)$ subtrees of node v, and subject to the flow in $G[v,j-1]$ having value α. This is maximized to determine $\theta(v,j,k)$.

Equation 2b ensures that the flow into node w is sufficient; recall that it need not have been sufficient in $G[w,d_w]$, and anyhow the arcs (v,w) and (w,v) are not in $G[w,d_w]$. The value contributed by the flow in $G[w,d_w]$ is constrained to be β. Since Equation 2b expresses the feasibility of the flow, the excess might as well be as large as possible, so $\theta(w,d_w,\beta)$ is assumed.

Equations 2c and 2d constrain the flow values on arcs (v,w) and (w,v). Equation 2e ensures that the four parts of the flow add up to k, and Equations 2f through 2h are needed to guarantee feasibility.

So the program specified by Equations 2 determines $\theta(v,j,k)$. The final value computed by the algorithm is $\theta(\rho,d_\rho,M)$. If this value is non-negative, then the necessity part of the proof of Lemma 5 shows that the value of x found by the algorithm is a solution to the instance $(G,f,"=",M)$. Similarly, if $\theta(\rho,d_\rho,M) < 0$, then Lemma 5 implies that the instance has no solution. Therefore the algorithm is correct.

Running time: The computation step $(*)$ is executed only $M \cdot |A|$ times. The rest of this proof shows that each execution can be solved in VPP time. This yields the conclusion that the algorithm runs in VPP time.

A call with $j = 0$ requires solution of the program expressed by Equations 1. For a node v, this can be solved with $O(\log(u_{0,v}+1))$ evaluations of $f_{0,v}(\cdot)$ by using binary search over possible values of $x_{0,v}$, because $f_{0,v} \in \mathcal{AP}(\omega^{1,=})$, and $f_{0,v}$ is therefore either monotonically increasing or monotonically decreasing.

A call with $j > 0$ requires solution of the program expressed by Equations 2. Only $O(k^3)$ distinct 4-tuples $(\alpha,\beta,\gamma,\delta)$ satisfy Equations 2e and 2h, so if the program can be solved for each 4-tuple in VPP time, then the entire algorithm will run within the allowed time.

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

$$\begin{aligned} & f_{v,w}(x_{v,w}) = \gamma \\ & x_{v,w} \leq u_{v,w} \\ & x_{v,w} \in \mathcal{Z}^+ \end{aligned} \tag{3}$$

Let $\lambda_{v,w}$ and $\xi_{v,w}$ be the smallest and largest values that satisfy Equations 3; they can be found with $O(\log(u_{v,w}+1))$ evaluations of $f_{v,w}(\cdot)$ by binary search, because $f_{v,w} \in \mathcal{AP}(\omega^{1,=})$. Similarly, the corresponding values of $\lambda_{w,v}$ and $\xi_{w,v}$ for $x_{w,v}$ can be found with $O(\log(u_{w,v}+1))$ evaluations of $f_{w,v}(\cdot)$. The following program yields the optimal values of $x_{v,w}$ and $x_{w,v}$, given $(\alpha, \beta, \gamma, \delta)$.

$$\begin{aligned} \max_{x_{v,w}, x_{w,v}} \quad & \mu_{w,v} x_{w,v} - x_{v,w} \\ \text{s.t.} \quad & \mu_{v,w} x_{v,w} - x_{w,v} \geq b_w - \theta(w, d_w, \beta) \\ & \lambda_{v,w} \leq x_{v,w} \leq \xi_{v,w} \\ & \lambda_{w,v} \leq x_{w,v} \leq \xi_{w,v} \\ & x_{v,w}, x_{w,v} \in \mathcal{Z}^+ \end{aligned} \tag{4}$$

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

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

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

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

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

3. $\underline{\mu_{v,w},\ \mu_{w,v} \geq 1}$: Suppose that, in some feasible solution of Equations 4, both $x_{v,w} < \xi_{v,w}$ and $x_{w,v} < \xi_{w,v}$. Incrementing each variable by one would yield a new feasible solution with objective value at least as good as the value of the previous solution. Therefore, attention can be restricted to solutions in which at least one of these variables is at its upper bound. Each case is considered, and the one with the larger objective function value is used.

   (a) $\underline{x_{v,w} = \xi_{v,w}}$: Set $x_{w,v}$ as in case 1.

   (b) $\underline{x_{w,v} = \xi_{w,v}}$: Set $x_{v,w}$ as in case 2.

■

Apply Lemma 4 and Lemma 3 in [SO94] to Theorem 3 in order to conclude that the optimization problem corresponding to $(\mathcal{C}', \mathcal{AP}(\omega^{1,=}), \omega^{1,=}, \text{Feas})$ has an FAS.

Corollary 1 *The optimization problem* $\left(\mathcal{C}', \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, Opt\right)$ *has an FAS.*

4.2 Using Reduction

The previous section presented an explicit VPP algorithm. This section uses reduction to demonstrate the existence of a VPP algorithm without actually writing out the algorithm. As a consequence, results about $\mathcal{C}'$ are generalized to $\mathcal{C}$.

Lemma 6 $\left(\mathcal{C}, \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, Feas\right) \equiv_{\text{VPP}} \left(\mathcal{C}', \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, Feas\right)$

Proof. Let $\Pi = \left(\mathcal{C}, \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Feas}\right)$ and $\Pi' = \left(\mathcal{C}', \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Feas}\right)$.

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

$\Pi \propto_{\text{VPP}} \Pi'$: Let $I = \left(S, f, \omega^{1,=}, M\right) \in \Pi$ be given, where S is defined by a graph $G = (N, A, u, \mu, b) \in \mathcal{C}$. Define $I' = \left(S', f', \omega^{1,=}, M\right) \in \Pi'$, where S' is defined by the graph $G' = (N', A', u', \mu', b') \in \mathcal{C}'$. The graph G' is the same as G, except as follows:

1. (Eliminate arcs into node 0) For each arc $a = (v, 0) \in A$:
   (a) Delete arc a from A'.
   (b) Add a node w_a to N' with $b_{w_a} = 0$.
   (c) Add an arc $a' = (v, w_a)$ to A' with $u_{a'} = u_a$, $\mu_{a'} = \mu_a$, and $f'_{1,a'}\left(x_{a'}\right) \equiv f_{1,a}\left(x_{a'}\right)$.
2. (Eliminate parallel arcs from node 0) For each node $v \in \{1, \ldots, m\}$, if the graph contains multiple arcs from node 0 to node v, do the following steps for all but one of the arcs. Let $a = (0, v)$ be the arc on which the steps are being performed.
   (a) Delete arc a from A'.
   (b) Add a node w_a to N' with $b_{w_a} = 0$.
   (c) Add an arc $a' = (0, w_a)$ to A' with $u_{a'} = u_a$, $\mu_{a'} = 1$, and $f'_{1,a'}\left(x_{a'}\right) \equiv 0$.
   (d) Add an arc $a'' = (w_a, v)$ to A' with $u_{a''} = u_a$, $\mu_{a''} = \mu_a$, and $f'_{1,a''}\left(x_{a''}\right) \equiv f_{1,a}\left(x_{a''}\right)$.
3. (Guarantee an arc from node 0 to each other node) For each node $v \in N$ such that arc $(0, v) \notin A$: Add an arc $a' = (0, v)$ to A' with $u_{a'} = 0$, $\mu_{a'} = 1$, and $f'_{1,a'}\left(x_{a'}\right) \equiv 0$.
4. (Guarantee pairs of anti-parallel arcs) For each arc $(v, w) \in A$ such that arc $(w, v) \notin A$: Add an arc $a' = (w, v)$ to A' with $u_{a'} = 0$, $\mu_{a'} = 1$, and $f'_{1,a'}\left(x_{a'}\right) \equiv 0$.
5. (Guarantee connectedness) If the subgraph of G induced by nodes $\{1, \ldots, n\}$ is not connected:
   (a) Add a node w to N' with $b_w = 0$.

(b) Add pairs of arcs $a' = (v, w)$ and $a'' = (w, v)$ to A', with $u_{a'} = u_{a''} = 0$, $\mu_{a'} = \mu_{a''} = 1$, and $f'_{1,a'}(x_{a'}) \equiv f'_{1,a''}(x_{a''}) \equiv 0$, to at least one node v in each component of the subgraph.

For each arc a that is in both A and A', define $f'_{1,a}(x_a) \equiv f_{1,a}(x_a)$; then $f' \in \mathcal{AP}(\omega^{1,=})$. By construction, $G' \in \mathcal{C}'$; furthermore, G' can be constructed in VPP time, and $L(I)$ is polynomial in $L(I')$. The reduction consists of constructing G' and solving I' as described in Section 4.1. This yields a solution of I directly using the correspondences described in the construction of G'. ■

Theorem 3 and Lemma 6 together with Lemma 4 in [SO94] yield the following corollary.

Corollary 2 *The feasibility problem* $(\mathcal{C}, \mathcal{AP}(\omega^{1,=}), \omega^{1,=}, Feas)$ *has a VPP algorithm and the optimization problem* $(\mathcal{C}, \mathcal{AP}(\omega^{1,=}), \omega^{1,=}, Opt)$ *has an FAS.*

Corollary 2 is a powerful building block. An explicit VPP algorithm need not be written down and verified in order to prove the existence of an FAS for another problem; instead, problems of interest can be VPP reduced to the problem $(\mathcal{C}, \mathcal{AP}(\omega^{1,=}), \omega^{1,=}, \text{Feas})$. This is done for several STT problems in Section 5. Several knapsack, job scheduling, and lot-sizing problems are treated this way in Sections 6 through 9.

5 Easy Problems and Hard Problems

Many problems can be defined on STT networks besides those discussed in the previous section. This section explores the boundary between "easy" and "hard" problems. The easy problems are those for which an FAS exists; the hard problems are those for which no VPP algorithm can exist unless $\mathcal{P} = \mathcal{NP}$. Section 5.1 examines problems with multiple criteria and Section 5.2 considers problems in which the flows must be exact. A summary of the results is tabulated in Figure 6 at the end of this section.

The proofs of hardness use the Subset Sum problem, which is $\mathcal{NP}$-complete[GJ79].

Subset Sum

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

Question: Find a set $S \subseteq \{1, \ldots, n\}$ such that $\sum_{j \in S} v_j = V$.

5.1 Multi-Criteria Objectives

This section extends the previous results about $\mathcal{C}$ to the multi-criteria case. Theorem 4 generalizes Corollary 2 to include multiple criteria and order-reversing functions. Corollary 3 uses Theorem 5 to achieve similar results for problems with deficient flows. Theorem 6 illustrates that problems with parallel arcs can be hard.

Theorem 4 *For each $r \in \mathcal{Z}^+$ and $\omega \in \Omega^r$, the problem $(\mathcal{C}, \mathcal{AM}(\omega), \omega, Opt)$ has an FAS.*

Proof. This follows from applying Lemma 4 and Lemma 2 in [SO94] to Corollary 2 and Theorem 2. ■

A similar theorem is true for problems with deficient flows; it will be proved as a corollary to the following theorem, which describes a more general relationship between problems with sufficient flows and those with deficient flows.

Theorem 5 *Let $A^* \in \{A1, AP\}$, $C^* \in \{C1, CG\}$, and $M^* \in \{M1, MG\}$. Define STT networks $\Psi = (A^*/C^*/M^*/NM/FS)$ and $\Psi' = (A^*/C^*/M^*/NM/FD)$. For any direction vector $\omega \in \Omega^r$, define the problems $\Pi = (\Psi, \mathcal{AM}(\omega), \omega, Feas)$ and $\Pi' = (\Psi', \mathcal{AM}(\omega), \omega, Feas)$. Then $\Pi \equiv_{\text{VPP}} \Pi'$.*

Proof. The case of $\Pi \propto_{\text{VPP}} \Pi'$ will be proved; the converse is proved similarly. Suppose that an instance $I = (S, f, \omega, M) \in \Pi$ is given, where S is defined by the graph $G = (N, A, u, \mu, b)$. Define the graph $G' = (N, A, u, \mu, b')$, where for $v \in N - \{0\}$,

$$b'_v = \sum_{\substack{a \in A: \\ a=(w,v)}} \mu_a u_a - \sum_{\substack{a \in A: \\ a=(v,w)}} u_a - b_v \ ,$$

and let S' be the feasible set defined by G'.

Define the function f' by $f'_i(x) = f_i(u - x)$, $i = 1, \ldots, r$. Since $f \in \mathcal{AM}(\omega)$, Lemmas 2 and 1 can be used to show that $f' \in \mathcal{AM}(\omega)$. So $I' = (S', f', \omega, M) \in \Pi'$.

Since $f(x) = f'(u - x)$, the proof will be completed by showing that a flow x is feasible for I if and only if the flow $(u - x)$ is feasible for I'. A flow x is feasible for I if and only if $0 \leq x \leq u$ and

$$\sum_{\substack{a \in A: \\ a=(w,v)}} \mu_a x_a - \sum_{\substack{a \in A: \\ a=(v,w)}} x_a - b_v \geq 0 \, . \tag{5}$$

But $x' = u - x$ satisfies $0 \leq x' \leq u$, and

$$\sum_{\substack{a \in A: \\ a=(w,v)}} \mu_a (u_a - x_a) - \sum_{\substack{a \in A: \\ a=(v,w)}} (u_a - x_a) - b'_v \leq 0$$

if and only if Equation 5 is true. This demonstrates the claim, so $\Pi \propto_{\text{VPP}} \Pi'$. ■

Note that the construction of b' requires that both supply and demand nodes be allowed, so the fourth component of the subclass definition in Theorem 5 is NM. The following corollary is the analog of Theorem 4 for problems with deficient flows. It follows from applying Lemma 3 and Lemma 4 in [SO94] to Theorem 5 and Theorem 4.

Corollary 3 *Let $\mathcal{C}_1 = (A1/CG/MG/NM/FD)$. Then for each $r \in \mathcal{Z}^+$ and $\omega \in \Omega^r$, the optimization problem $(\mathcal{C}_1, \mathcal{AM}(\omega), \omega, Opt)$ has an FAS.*

Theorem 4 and Corollary 3 cannot, however, be extended to cover STT networks that contain parallel arcs, as shown by the following theorem.

Theorem 6 *Let* $\mathcal{C}_1 = (AP/CG/M1/N0/FS)$ *and* $\mathcal{C}_2 = (AP/CG/M1/N0/FD)$. *Neither the problem* $\left(\mathcal{C}_1, \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, Feas\right)$ *nor the problem* $\left(\mathcal{C}_2, \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, Feas\right)$ *has a VPP algorithm unless* $\mathcal{P} = \mathcal{NP}$.

Proof. The theorem will be proved by reducing Subset Sum to each of the STT problems in such a way that a VPP algorithm for one of the latter would yield a polynomial-time algorithm for the former. Since Subset Sum in $\mathcal{NP}$-complete, such a VPP algorithm cannot exist unless $\mathcal{P} = \mathcal{NP}$.

First consider the problem $\Pi_1 = \left(\mathcal{C}_1, \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Feas}\right)$. Given an instance (n, V, v) of Subset Sum, construct an STT network $G = (N, A, u, \mu, b)$ as follows. The graph is pictured in Figure 3. The set N has three nodes, and both non-source nodes are transshipment nodes; that is, $N = \{0, 1, 2\}$, with $b_1 = b_2 = 0$. The arcs are described in the following table:

j	Arc	Capacity	Multiplier	Value $f_j(x_j)$
$1, \ldots, n$	$(1,2)$	v_j	1	$2n+2$ if $x_j = v_j$ 2 if $0 < x_j < v_j$ 0 if $x_j = 0$
$n+1$	$(0,1)$	V	1	0
$n+2$	$(2,0)$	V	1	1 if $x_j = V$ 0 o.w.

The objective function $f(x) = \sum_{i=1}^{n+2} f_j(x_j)$ is in $\mathcal{AP}\left(\omega^{1,=}\right)$. It will be shown that the instance of Subset Sum has a solution if and only if at least one instance $\left(G, f, \omega^{1,=}, k(2n+2)+1\right)$, $k = 0, \ldots, n$, of Π_1 has a solution, and that the solution to any of these instances immediately yields a solution to the instance of Subset Sum. The reduction of the instance of Subset Sum to the instance of Π_1 is polynomial in the size of the Subset Sum instance, so a VPP algorithm for Π_1 yields a polynomial-time algorithm for Subset Sum.

If the Subset Sum instance has a solution: Suppose that $S \subseteq \{1, \ldots, n\}$, with $|S| = \overline{k}$, is a solution to the instance of Subset Sum. One feasible solution to the constructed instance of Π_1 with $k = \overline{k}$ is $x_{0,1} = V$, $x_{2,0} = V$, $x_j = v_j$ for $j \in S$, and $x_j = 0$ for $j \in \{1, \ldots, n\} - S$. This solution has value $(\overline{k}(2n+2)+1)$.

If the instance of Π_1 has a solution: Suppose that x is a solution to the instance of Π_1 with value $(\overline{k}(2n+2)+1)$. Since this value is odd, the flow in arc $(2,0)$ must be V; this leaves $\overline{k}(2n+2)$ of the objective function unaccounted for.

Since the flow into node 2 is sufficient, at least V units of flow must enter node 2 on arcs 1 through n. Since the flow into node 1 is sufficient, and since $u_{0,1} = V$, the total flow on arcs 1 through n must be exactly V. Since the remaining part of the objective function is divisible by (2n+2), each of the arcs 1 through n with

non-zero flow must be used to capacity; no subset of partially-full arcs could contribute a factor of (2n+2) to the objective function. The indices of the arcs 1 through n with non-zero flow constitute a solution of size $\overline{k}$ to the instance of Subset Sum.

The second part of the theorem is about the problem $\left(\mathcal{C}_2, \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Feas}\right)$. The proof is similar, except that the objective functions on arcs $n+1$ and $n+2$ are interchanged. ■

5.2 Exact Flows

The graphs discussed so far must have either sufficient or deficient flows. This section considers problems in which the flows must be exact, that is, the excess flow into each node must be zero. Theorem 7 identifies some such problems that can be solved by an FAS. Theorems 8 and 9 show that Theorem 7 cannot be extended to graphs with both supply and demand nodes or to problems with objective functions that are not order-preserving.

Theorem 7 *Let* $\mathcal{C}_1 = (A1/CG/M1/ND/FE)$ *and* $\mathcal{C}_2 = (A1/CG/M1/NS/FE)$. *For any direction vector* ω, *an FAS exists for each of the problems* $(\mathcal{C}_1, \mathcal{AP}(\omega), \omega, Opt)$ *and* $(\mathcal{C}_2, \mathcal{AP}(\omega), \omega, Opt)$.

Proof. First consider the problem $\Pi_1 = (\mathcal{C}_1, \mathcal{AP}(\omega), \omega, \text{Opt})$. Let r be the number of components in ω. Define the STT network feasible set $\mathcal{C}_3 = (A1/CG/M1/ND/FS)$ and the problem $\Pi_3 = (\mathcal{C}_3, \mathcal{AP}(\omega), \omega, \text{Feas})$. Note that $\mathcal{C}_3 \subset \mathcal{C}$, so Π_3 has an FAS.

The reduction $\Pi_1 \propto_{\text{VPP}} \Pi_3$ will be proved. By Lemma 4 in [SO94], this will mean that Π_1 has an FAS, thereby proving the first half of the theorem. The proof for $(\mathcal{C}_2, \mathcal{AP}(\omega), \omega, \text{Opt})$ is substantially the same, and so will not be presented; it uses a reduction to a problem defined on the feasible set $(A1/CG/M1/NS/FD)$.

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

Consider any flow x that solves I_3. This flow can be decomposed into flows around circuits and flows on chains (simple directed paths) from supply nodes to demand nodes[FF62, AMO93]. Suppose that the flow around some circuit in such a decomposition is non-zero; since f is order-preserving w.r.t. ω, no component of the objective would be made worse by eliminating the flow around the circuit. So the circuits in the decomposition can be assumed to have zero flow.

All nodes except node 0 are demand nodes, so all chains in the decomposition are from node 0 to another node. For each node v with positive excess, that is, $e_v(x) > 0$, consider the chains from node 0 to node v. Reducing the flow on one or more of these chains yields a flow x' with $e_v(x') = 0$. As with the circuits, no component of the objective function will be made worse by reducing the flow in this way.

Reduce the flow in this way for each node to obtain an exact flow $\overline{x}$. The flow $\overline{x}$ solves I_1, so $\Pi_1 \propto_{\text{VPP}} \Pi_3$. ■

The next theorem shows that requiring an exact flow in a network that contains both supply and demand nodes results in a hard problem.

Theorem 8 *Let $\mathcal{C}_1 = (A1/CG/M1/NM/FE)$ be an STT network. Then no VPP algorithm exists for the problem $\left(\mathcal{C}_1, \mathcal{AP}\left(\omega^{1,<}\right), \omega^{1,<}, Feas\right)$ unless $\mathcal{P} = \mathcal{NP}$.*

Proof. The theorem will be proved by reducing Subset Sum to the problem $\Pi = \left(\mathcal{C}_1, \mathcal{AP}\left(\omega^{1,<}\right), \omega^{1,<}, \text{Feas}\right)$ in such a way that a VPP algorithm for Π would directly yield a polynomial-time algorithm for Subset Sum. Since the latter problem is $\mathcal{NP}$-complete, however, such a VPP algorithm cannot exist unless $\mathcal{P} = \mathcal{NP}$.

Given an instance (n, V, v) of Subset Sum, an STT network is constructed, as illustrated in Figure 4. Intuitively, the constructed network has a node with supply v_j for each item j. An $(n+1)$-st node with demand V represents the desired sum. The arcs used to satisfy the demand define a solution to the instance of Subset Sum.

More formally, construct an STT network $G = (N, A, u, \mu, b)$ as follows. The set N has $n+2$ nodes $\{0, 1, \ldots, n, n+1\}$. Node j has supply v_j, that is, $b_j = -v_j$, $j = 1, \ldots, n$. Node $n+1$ has demand V, that is, $b_{n+1} = V$. The set A has $2n$ arcs; they are $\alpha_j = (j, 0)$ and $\beta_j = (j, n+1)$ for $j = 1, \ldots, n$. Arcs α_j and β_j have capacity v_j, multiplier 1, and the following objective function:

$$g(x_k) = \begin{cases} 1 & \text{if } x_k \geq 1 \\ 0 & \text{if } x_k = 0 \end{cases}.$$

The objective function $f(x) = \sum_{j=1}^{n} \left[g\left(x_{\alpha_j}\right) + g\left(x_{\beta_j}\right)\right]$ is in $\mathcal{AP}\left(\omega^{1,<}\right)$. It will be shown that the instance of Subset Sum has a solution if and only if the instance $\left(G, f, \omega^{1,<}, n\right)$ of Π has a solution, and that the solution to this instance immediately yields the solution to the instance of Subset Sum. The reduction of the instance of Subset Sum to the instance of Π is polynomial in the size of the Subset Sum instance, so a VPP algorithm for Π yields a polynomial-time algorithm for Subset Sum.

If the Subset Sum instance has a solution: Suppose that $S \subseteq \{1, \ldots, n\}$ is a solution to the instance of Subset Sum. One feasible solution to the constructed instance of Π is $x_{\beta_j} = v_j$ for $j \in S$, $x_{\beta_j} = 0$ for $j \notin S$, and $x_{\alpha_j} = v_j - x_{\beta_j}$, $j = 1, \ldots, n$. This solution has value n.

If the instance of Π has a solution: Suppose that x is a solution to the instance of Π with value not greater than n. Each node j must dispose of its supply on arc α_j, β_j, or both. The objective function must therefore include at least unit value for the flow leaving each of these nodes, and hence must be at least n.

The solution to Π therefore has value exactly n, which means that each of the nodes $1, \ldots, n$ uses exactly one of its outgoing arcs. Let S be the index set of the arcs β_j that have non-zero flow. Then for $j \in S$, it must be the case that $x_{\beta_j} = v_j$, and so $\sum_{j \in S} x_{\beta_j} = V$. So S constitutes a solution to the instance of Subset

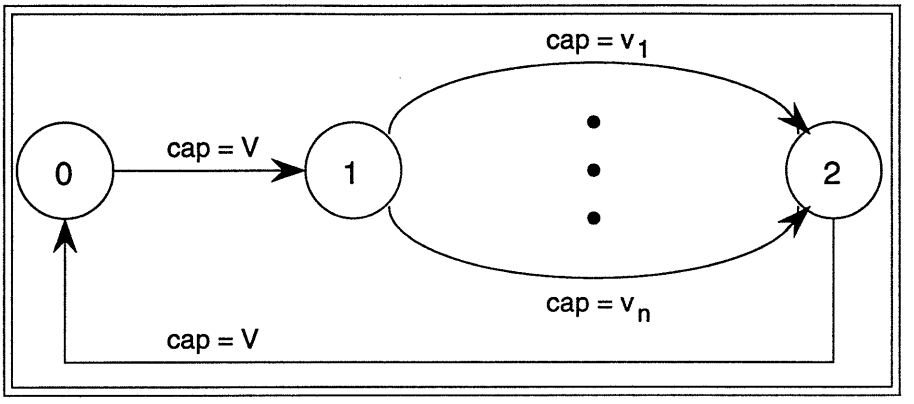

Figure 3: The graph constructed in the proof of Theorem 6.

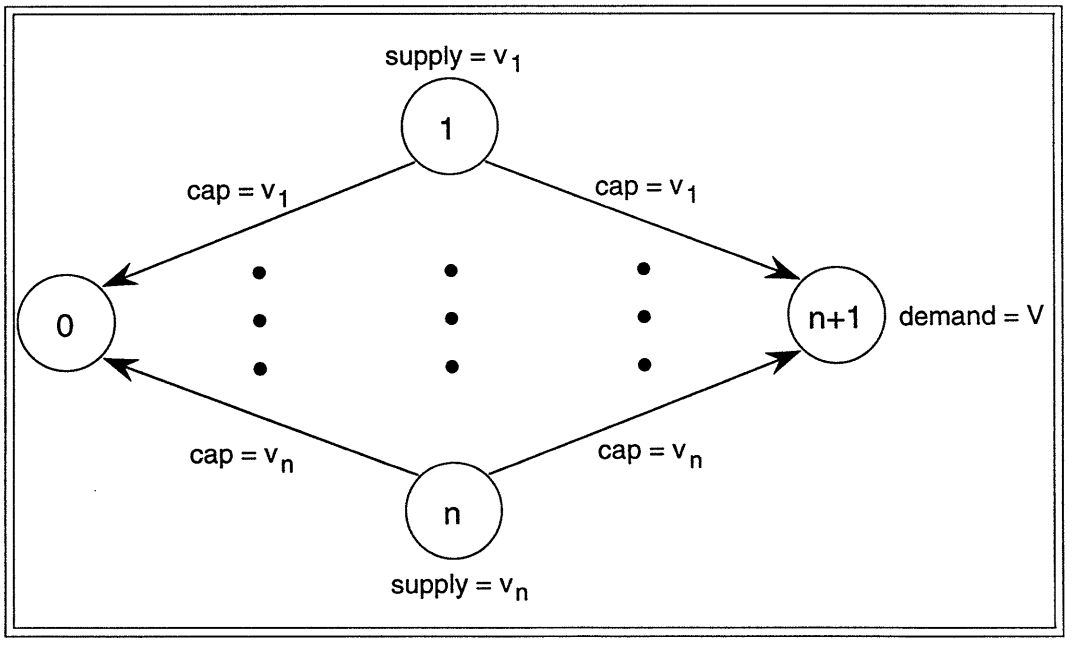

Figure 4: The graph constructed in the proof of Theorem 8.

Sum. ■

Similarly, the next theorem shows that using objective functions that are not order-preserving on networks that contain only transshipment nodes results in a hard problem.

Theorem 9 *Let* $\mathcal{C}_1 = (A1/CG/M1/N0/FE)$ *be an STT network. Then no VPP algorithm exists for the problem* $\left(\mathcal{C}_1, \mathcal{AP}\left(\omega^{1,<}\right), \omega^{1,>}, \textit{Feas}\right)$ *unless* $\mathcal{P} = \mathcal{NP}$.

Proof. The theorem will be proved by reducing Subset Sum to the problem $\Pi = \left(\mathcal{C}_1, \mathcal{AP}\left(\omega^{1,<}\right), \omega^{1,>}, \text{Feas}\right)$ in such a way that a VPP algorithm for Π would directly yield a polynomial-time algorithm for Subset Sum. Since the latter problem is $\mathcal{NP}$-complete, however, such a VPP algorithm cannot exist unless $\mathcal{P} = \mathcal{NP}$.

Given an instance (n, V, v) of Subset Sum, an STT network is constructed, as pictured in Figure 5. Intuitively, each node j receives flow v_j from node 0 and either sends it on to node $(n+1)$ or returns it to node 0. The arcs into node $n+1$ correspond to a solution for the instance of Subset Sum.

More formally, construct an STT network $G = (N, A, u, \mu, b)$ as follows. The set N has $n+2$ nodes $\{0, 1, \ldots, n, n+1\}$. All nodes are transshipment nodes, that is, $b_j = 0$, $j = 1, \ldots, n+1$. The set A has $3n+1$ arcs that are described in the following table:

Arc		Capacity	Multiplier	Value $f_k(x_k)$
$(0, j)$,	$j = 1, \ldots, n$	v_j	1	0
$(j, 0)$,	$j = 1, \ldots, n$	v_j	1	1 if $x_k = v_j$
$(j, n+1)$,	$j = 1, \ldots, n$			0 o.w.
$(n+1, 0)$		V	1	1 if $x_k = V$
				0 o.w.

The objective function $f(x) = \sum_{k \in A} f_k(x_k)$ is in $\mathcal{AP}\left(\omega^{1,<}\right)$. It will be shown that the instance of Subset Sum has a solution if and only if the instance $\left(G, f, \omega^{1,>}, n+1\right)$ of Π has a solution, and that the solution to this instance immediately yields the solution to the instance of Subset Sum. The reduction of the instance of Subset Sum to the instance of Π is polynomial in the size of the Subset Sum instance, so a VPP algorithm for Π yields a polynomial-time algorithm for Subset Sum.

If the Subset Sum instance has a solution: Suppose that $S \subseteq \{1, \ldots, n\}$ is a solution to the instance of Subset Sum. One feasible solution to the constructed instance of Π is $x_{j,n+1} = v_j$ for $j \in S$ and $x_{j,n+1} = 0$ for $j \notin S$; $x_{n+1,0} = V$; and for $j = 1, \ldots, n$, $x_{0,j} = v_j$ and $x_{j,0} = v_j - x_{j,n+1}$. This solution has value $n+1$, and so solves the instance of Π.

If the instance of Π has a solution: Suppose that x is a solution to the instance of Π with value at least $n+1$. It must be the case that $x_{n+1,0} = V$ and that each node j, $j = 1, \ldots, n$, sends flow v_j in exactly one of the arcs $(j, n+1)$ and $(j, 0)$ and zero flow in the other. In order for this to happen, the flow in each arc $(0, j)$, $j = 1, \ldots, n$, must be v_j.

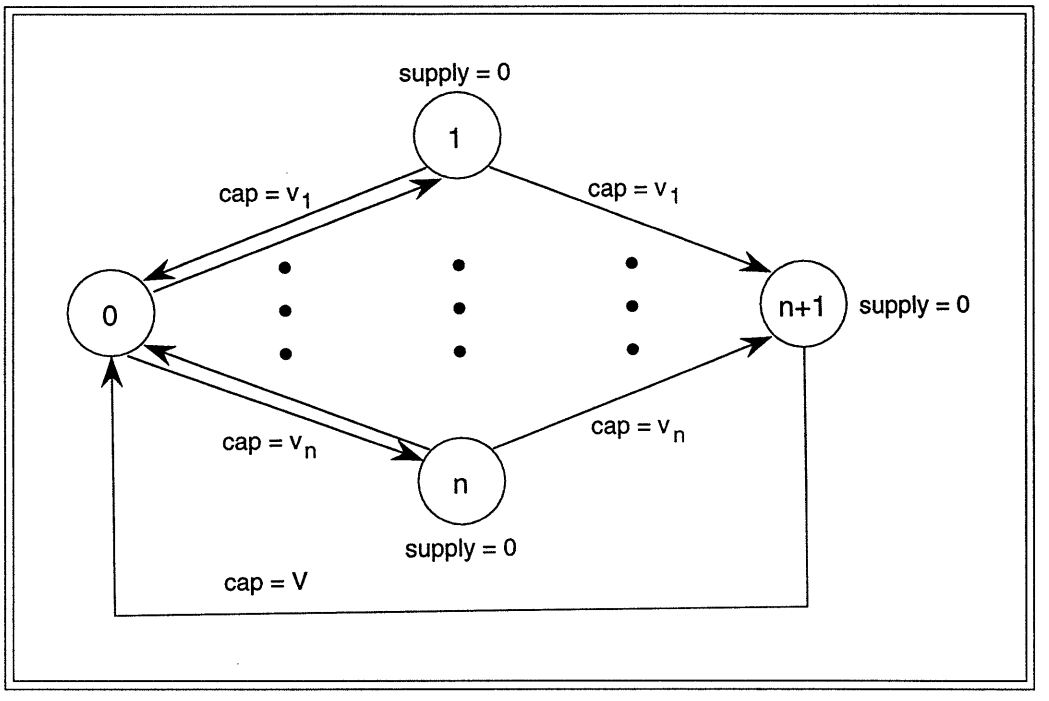

Figure 5: The graph constructed in the proof of Theorem 9.

$\mathcal{C}_1$	Π	Difficulty	Reference
$(A1/CG/MG/NM/FS)$	$(\mathcal{C}_1, \mathcal{AM}(\omega^{r,=}), \omega^{r,=}, \text{Opt})$	Easy	Theorem 4
$(A1/CG/MG/NM/FD)$	$(\mathcal{C}_1, \mathcal{AM}(\omega), \omega, \text{Opt})$	Easy	Corollary 3
$(AP/CG/M1/N0/FS)$	$(\mathcal{C}_1, \mathcal{AP}(\omega^{1,=}), \omega^{1,=}, \text{Feas})$	Hard	Theorem 6
$(AP/CG/M1/N0/FD)$	$(\mathcal{C}_1, \mathcal{AP}(\omega^{1,=}), \omega^{1,=}, \text{Feas})$	Hard	Theorem 6
$(A1/CG/M1/ND/FE)$	$(\mathcal{C}_1, \mathcal{AP}(\omega), \omega, \text{Opt})$	Easy	Theorem 7
$(A1/CG/M1/NS/FE)$	$(\mathcal{C}_1, \mathcal{AP}(\omega), \omega, \text{Opt})$	Easy	Theorem 7
$(A1/CG/M1/NM/FE)$	$(\mathcal{C}_1, \mathcal{AP}(\omega^{1,<}), \omega^{1,<}, \text{Feas})$	Hard	Theorem 8
$(A1/CG/M1/N0/FE)$	$(\mathcal{C}_1, \mathcal{AP}(\omega^{1,<}), \omega^{1,>}, \text{Feas})$	Hard	Theorem 9

Figure 6: Summary of STT complexity results.

Let S be the set of nodes j, $j = 1, \ldots, n$, that send non-zero flow in arc $(j, n+1)$. Since $b_{n+1} = 0$ and the excess at each node must be 0, it follows that $\sum_{j \in S} v_j = V$; so S constitutes a solution to the instance of Subset Sum. ■

The results of this section are summarized in Figure 6.

6 Knapsack Covering Problems

This section discusses the form of constraints that define the feasible regions for knapsack problems, as a prelude to the discussion of specific problems in the following two sections. The knapsack problems considered in this paper are special cases of the Multi-Dimensional Knapsack problem, which is formulated below. In general, no FAS can exist for this problem unless $\mathcal{P} = \mathcal{NP}$.

Integer Multi-Dimensional Knapsack Feasible Set

Instance: (n, m, V, v, u). The number of items n and number of constraints m are non-negative integers. The constraint upper bounds V_i, $i = 1, \ldots, m$; the item values $v_{i,j}$, $i = 1, \ldots, m$, $j = 1, \ldots, n$; and the item quantity upper bounds u_j, $j = 1, \ldots, n$; are non-negative integers such that $v_{i,j} \leq V_i$, $i = 1, \ldots, m$, $j = 1, \ldots, n$.

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

$$\begin{aligned} \sum_{j=1}^{n} v_j x_j &\leq V_i, \quad i = 1, \ldots, m \\ x_j &\leq u_j, \quad j = 1, \ldots, n \end{aligned}$$

This generic knapsack problem has "$\leq$" inequality constraints. This is characteristic of the "packing" form of knapsack problems, in that upper limits are placed on the amounts that can be packed into a knapsack. Many interesting problems, however, have feasible regions that are naturally expressed by knapsack constraints with "$\geq$" inequalities; these are called knapsack "covering" problems.

Definition 7 Let Ψ be a family of feasible regions such that each $S \in \Psi$ can be described as the set of all $x \in \mathcal{Z}^{n+}$ that satisfy, for some integers n, m_1, m_2; V_i, $i = 1, \ldots, m_1$; v_{ij}, $i = 1, \ldots, m_1$, $j = 1, \ldots, n$; W_i, $i = 1, \ldots, m_2$; w_{ij}, $i = 1, \ldots, m_2$, $j = 1, \ldots, n$; and u_j, $j = 1, \ldots, n$;

$$\begin{array}{rcll} \sum_{j=1}^{n} v_{ij} x_j & \leq & V_i, & i = 1, \ldots, m_1 \\ \sum_{j=1}^{n} w_{ij} x_j & = & W_i, & i = 1, \ldots, m_2 \\ x_j & \leq & u_j, & j = 1, \ldots, n \, . \end{array} \tag{6}$$

Such a set S is said to be of the *packing* type. The *covering form* of S, written S^{cov}, is the feasible region defined as the set of all $y \in \mathcal{Z}^{n+}$ that satisfy

$$\begin{array}{rcll} \sum_{j=1}^{n} v_{ij} y_j & \geq & \sum_{j=1}^{n} v_{ij} u_j - V_i, & i = 1, \ldots, m_1 \\ \sum_{j=1}^{n} w_{ij} y_j & = & \sum_{j=1}^{n} w_{ij} u_j - W_i, & i = 1, \ldots, m_2 \\ y_j & \leq & u_j, & j = 1, \ldots, n \, , \end{array}$$

and the covering form of Ψ is

$$\Psi^{cov} = \{ S^{cov} : S \in \Psi \} .$$

■

The present work primarily considers knapsack problems of the packing form. The following theorem shows that this focus typically does not sacrifice much generality.

Theorem 10 *Let Ψ be a family of feasible regions of the packing type and $\omega \in \Omega$. Then*

$$\begin{array}{lll} (1) & (\Psi, \mathcal{AP}(\omega), \omega, Opt) & \equiv_{\text{VPP}} \; (\Psi^{cov}, \mathcal{AR}(\omega), \omega, Opt) . \\ (2) & (\Psi, \mathcal{AR}(\omega), \omega, Opt) & \equiv_{\text{VPP}} \; (\Psi^{cov}, \mathcal{AP}(\omega), \omega, Opt) . \\ (3) & (\Psi, \mathcal{AM}(\omega), \omega, Opt) & \equiv_{\text{VPP}} \; (\Psi^{cov}, \mathcal{AM}(\omega), \omega, Opt) . \end{array}$$

Proof. Statement (3) follows directly from statements (1) and (2). The proof that $(\Psi, \mathcal{AP}(\omega), \omega, \text{Opt}) \propto_{\text{VPP}} (\Psi^{cov}, \mathcal{AR}(\omega), \omega, \text{Opt})$ is given here; the remaining three reductions in (1) and (2) are proved similarly.

Let $(S, f, \omega) \in (\Psi, \mathcal{AP}(\omega), \omega, \text{Opt})$. For any $x \in S$, let $y = u - x$. Then $y \in S^{cov}$; for example,

$$\begin{array}{rcl} \sum_{j=1}^{n} v_{ij} y_j & = & \sum_{j=1}^{n} v_{ij} (u_j - x_j) \\ & \geq & \sum_{j=1}^{n} v_{ij} u_j - V_i \, . \end{array}$$

Now define the function $f'(y) = f(u - y)$. Then $f' \in \mathcal{AR}(\omega)$: if $y_1, y_2 \in S^{cov}$, then

$$\begin{aligned} y_1 \leq y_2 &\Rightarrow u - y_2 \leq u - y_1 \\ &\Rightarrow D_\omega(f(u - y_2), f(u - y_1)) \qquad \text{because } f \in \mathcal{AP}(\omega) \\ &\Rightarrow D_\omega(f'(y_2), f'(y_1)) \ . \end{aligned}$$

Furthermore, $f'(u - x) = f(x)$. So $(\Psi, \mathcal{AP}(\omega), \omega, \text{Opt}) \propto_{\text{VPP}} (\Psi^{cov}, \mathcal{AR}(\omega), \omega, \text{Opt})$. ■

Many kinds of problems admit an FAS if the feasible region has only packing constraints or only covering constraints. Combining the two kinds of constraints, however, often yields problems that have no FAS unless $\mathcal{P} = \mathcal{NP}$. That is why knapsack problems in which the knapsack constraint is an equality are generally difficult to solve.

For some problems, such as the Partially Ordered Knapsack problem discussed in Section 8, not all the constraints are reversed to define the covering form of the problem. Theorem 10 cannot be directly applied to such problems, but it can sometimes be used indirectly, as it is in Section 8.

7 The Arborescent Knapsack Problem

This section and the next discuss the existence and non-existence of FAS's for a variety of knapsack and scheduling problems. Note that the results presented are existential. The characteristics of a particular problem must be considered carefully in order to develop a practical algorithm for its solution. An excellent example of such an analysis appears in [Law79].

In the Arborescent Knapsack problem, items are grouped into sub-knapsacks, or stuffsacks. Stuffsacks may be nested within other stuffsacks and each stuffsack has its own capacity bound. This structure is called "arborescent" because the "nested within" relationship on the stuffsacks can be represented by a directed tree, or arborescence, as illustrated in Example 2 later in this section. After defining the Arborescent Knapsack problem and proving that it has an FAS, two special cases that arise in the context of job scheduling are discussed.

7.1 The General Problem

After formalizing the notion of stuffsacks, the Integer Arborescent Knapsack feasible region is defined and the existence of FAS's for certain problems defined on it is proved.

Definition 8 A collection of sets $\mathcal{B} = \{B_1, \ldots, B_m\}$ is called *arborescent* if for each pair $i, j \in \{1, \ldots, m\}$, with $i \neq j$, one of the following relations holds: $B_i \subset B_j$, $B_j \subset B_i$, or $B_i \cap B_j = \emptyset$. $\mathcal{B}$ is sometimes called a *collection of stuffsacks.* ■

The Arborescent Knapsack feasible set may now be defined formally as follows:

Integer Arborescent Knapsack Feasible Set

Instance: $(n, m, \mathcal{B}, V, v, u)$. The number of items n and number of stuffsacks m are non-negative integers. The collection of stuffsacks $\mathcal{B}$ is an arborescent collection of m sets $B_i \subseteq \{1, \ldots, n\}$, $i = 1, \ldots, m$, with $B_m = \{1, \ldots, n\}$. The stuffsack volumes V_i, $i = 1, \ldots, m$; 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 if $j \in B_i$, then $v_j \leq V_i$, $i = 1, \ldots, m$, $j = 1, \ldots, n$.

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

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

The nesting property of the sets B_j gives the multiple constraints a special structure that allows the Arborescent Knapsack problem to have an FAS, despite the difficulty of the Multi-Dimensional Knapsack problem. An FAS that uses $O\left(n^3/\varepsilon^2\right)$ time for the binary version of this problem with a single linear, additively separable criterion is presented in [GL79b]. The following theorem generalizes their result to the integer case, in which an item may appear more than once in the knapsack (up to u_j copies of item j may be used). In addition, this theorem accommodates multiple criteria.

Theorem 11 *Let Ψ be the collection of Integer Arborescent Knapsack feasible sets and $\omega \in \Omega$. Then $(\Psi, \mathcal{AM}(\omega), \omega, Opt)$ has an FAS.*

Proof. Let $\Pi_1 = (\Psi, \mathcal{AM}(\omega), \omega, \text{Feas})$ and $\Pi_2 = (\mathcal{C}_2, \mathcal{AM}(\omega), \omega, \text{Feas})$, where the feasible set $\mathcal{C}_2 = (A1/CG/MG/N0/FD)$. The reduction $\Pi_1 \propto_{\text{VPP}} \Pi_2$ will be shown. By Lemma 4 in [SO94] and Corollary 3, this reduction will prove the present theorem.

Suppose that an instance $I_1 = (S, f, \omega, M)$ of Π_1 is given, with $S = S(n, m, \mathcal{B}, V, v, u)$. Assume that the sets B_i are "topologically" ordered, so that $B_i \subset B_k$ implies that $i < k$; such an ordering can be found quickly[AMO93]. Define the "first containing set" function ϕ for *items* by

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

and a similar function σ for *sets* by

$$\sigma(i) = \begin{cases} \min\{\, k : B_i \subset B_k \,\} & i = 1, \ldots, m-1 \\ 0 & i = m \, . \end{cases}$$

Define an instance $I_2 = (C_2, f', \omega, M)$ of Π_2, where C_2 is defined by an STT network $G = (N, A, u, \mu, b)$. An example of the following construction is given in Example 2. The set N has $(n + m + 1)$ nodes: a node α_j corresponding to each item j, a node β_i corresponding to each stuffsack i, and node 0. All nodes are

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

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

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

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

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

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

$$x_j = y_{0,\alpha_j},\ j = 1, \ldots, n\ . \tag{7}$$

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

$$\begin{aligned}
i = 1: \quad \sum_{j \in B_1} v_j x_j &= \sum_{j \in B_1} v_j y_{0,\alpha_j} && \text{by Equation 7} \\
&\leq \sum_{j \in B_1} y_{\alpha_j, \beta_{\phi(j)}} && \text{nodes have deficient excess} \\
&= \sum_{j \in B_1} y_{\alpha_j, \beta_1} && \text{sets are topologically ordered} \\
&\leq y_{\beta_1, \beta_{\sigma(1)}} && \text{node } \beta_1 \text{ has deficient excess.}
\end{aligned}$$

$$\begin{aligned}
i > 1: \quad \sum_{j \in B_i} v_j x_j &= \sum_{j:\phi(j)=i} v_j x_j + \sum_{k:\sigma(k)=i} \sum_{j \in B_k} v_j x_j && \text{arborescent network structure} \\
&\leq \sum_{j:\phi(j)=i} y_{\alpha_j, \beta_i} + \sum_{k:\sigma(k)=i} \sum_{j \in B_k} v_j x_j && \text{nodes have deficient excess} \\
&\leq \sum_{j:\phi(j)=i} y_{\alpha_j, \beta_i} + \sum_{k:\sigma(k)=i} \sum_{j \in B_k} y_{\beta_k, \beta_i} && \text{by induction} \\
&\leq y_{\beta_i, \beta_{\sigma(i)}} && \text{node } \beta_i \text{ has deficient excess.}
\end{aligned}$$

■

Example 2 (Network construction for the proof of Theorem 11) Let $n = 7$ and $m = 5$, with the following arborescent collection of sets:

$$\begin{aligned} B_1 &= \{3,4,5\} \\ B_2 &= \{6\} \\ B_3 &= \{1,2\} \\ B_4 &= \{1,2,3,4,5\} \\ B_5 &= \{1,2,3,4,5,6,7\}. \end{aligned}$$

Then the functions $\phi(\cdot)$ and $\sigma(\cdot)$ are defined as follows:

Item j	Set $\phi(j)$	Set i	Set $\sigma(i)$
1	3	1	4
2	3	2	5
3	1	3	4
4	1	4	5
5	1	5	0
6	2		
7	5		

and the corresponding network is shown in Figure 7. ■

7.2 The Maximum Job Sequencing with Due Dates Problem

A special case of the Arborescent Knapsack problem is the Triangular Knapsack problem, in which the stuffsacks are nested sequentially. More precisely, the Triangular Knapsack problem is an Arborescent Knapsack problem in which $m = n$ and $B_i = \{1, \ldots, i\}$. One reason for interest in this problem is that a common scheduling problem can be formulated as a Triangular Knapsack problem. Details about related scheduling problems can be found in [GLLRK79].

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

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

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

Question: Schedule the jobs to earn the maximum possible profit, where job j earns profit p_j if the job is completed before time d_j and earns 0 profit otherwise.

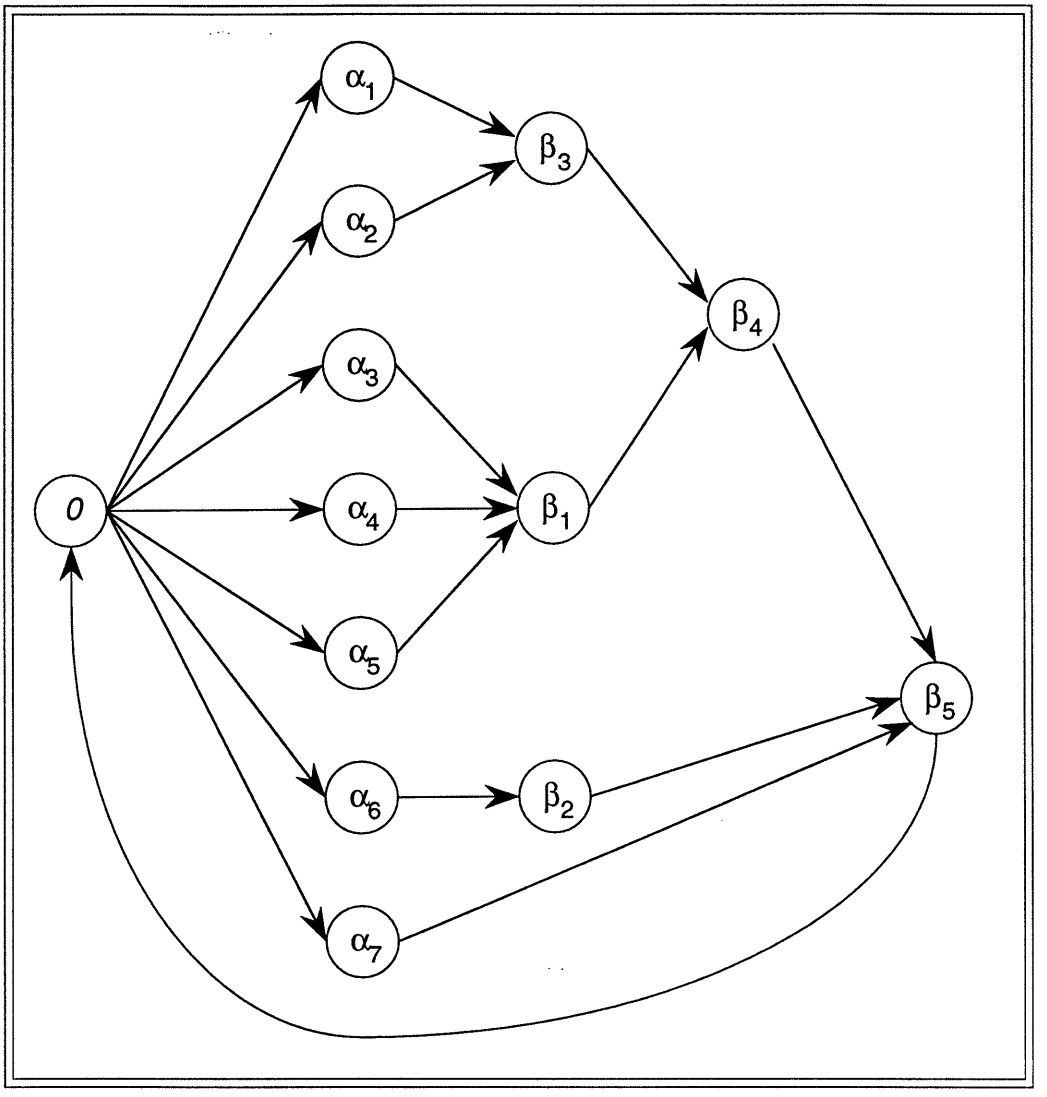

Figure 7: Network constructed as in the proof of Theorem 11 for the instance of Example 2.

An FAS that runs in time $O\left(n^2/\varepsilon\right)$ for this scheduling problem is described in [Sah76, Sah77].

To see that this problem can be formulated as a Triangular Knapsack problem, recall that the profit is independent of the order in which the jobs are performed, and depends only on the relationship of the finish times to the due dates. The set of optimal schedules must therefore include at least one schedule in which all the jobs that are completed on time are performed before those that are finished late; so the search for an optimal schedule can be restricted to schedules of this kind. Define the variables x_j, $j = 1, \ldots, n$, by

$$x_j = \begin{cases} 1 & \text{if job } j \text{ is completed on time,} \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

The Max JSD problem can then be formulated as:

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

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

Question: Find x to solve

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} p_j x_j & & \\ \text{s.t.} \quad & \sum_{k=1}^{j} t_k x_k \leq d_j, & & j = 1, \ldots, n \\ & x_j \in \{0, 1\}, & & j = 1, \ldots, n \end{aligned}$$

This has the form of a Triangular Knapsack problem. This formulation is similar to that alluded to in [Sah76, Sah77, GL79a] and described more fully in [Rot64, LM69, GL78, GL81] for a similar scheduling problem that is discussed in the next section. The results of Section 7.1 imply the existence of an FAS for the multi-criteria form of this problem.

7.3 The Minimum Job Sequencing with Due Dates Problem

The Job Sequencing with Due Dates Problem can be approached differently by assessing penalties for jobs that are completed late, rather than providing profits for jobs that are completed on time. The objective in this case is to minimize the sum of the penalties, and the problem is called the Min Job Sequencing with Due Dates (Min JSD) problem. An exponential-time dynamic programming algorithm for this problem is presented in [LM69]. An FAS that requires $O\left(n^3/\varepsilon\right)$ time is described in [GL78]; better algorithms that run in time $O\left(n^2 \log(n) + n^2/\varepsilon\right)$ are given in [GL79a, GL81].

The Min JSD problem is not just a trivial variant of the Max JSD problem, because the complexities of approximating the solution of a minimization problem and of the corresponding maximization problem can be quite different. It has been speculated that approximating solutions for the Min JSD problem may be

inherently harder than doing so for the Max JSD problem[GL81]. Nevertheless, the approximation schemes of [GL78, GL79a, GL81] are derived by reducing the Min JSD problem to the Min Arborescent Knapsack problem. The variables are the complements of those used in Equation 8:

$$y_j = \begin{cases} 1 & \text{if job } j \text{ is completed late,} \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

The problem can be formulated as follows:

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

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

Question: Find y to solve

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} p_j y_j \\ \text{s.t.} \quad & \sum_{k=1}^{j} t_k(1-y_k) \leq d_j, \qquad j = 1, \ldots, n \\ & y_j \in \{0,1\}, \quad j = 1, \ldots, n\ , \end{aligned}$$

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

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

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

Question: Find y to solve

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} p_j y_j \\ \text{s.t.} \quad & \sum_{k=1}^{j} t_k y_k \geq d'_j, \qquad j = 1, \ldots, n \\ & y_j \in \{0,1\}, \quad j = 1, \ldots, n, \end{aligned}$$

This is a Binary Arborescent Knapsack Covering problem. To our knowledge the Arborescent Knapsack Covering problem has only been treated by considering the special case of this scheduling problem. It does, however, satisfy the preconditions of Theorem 10. The previously known results for the Min JSD problem can therefore be generalized to guarantee an FAS for the Integer Arborescent Knapsack Covering problem with multiple criteria.

8 The Partially Ordered Knapsack Problem

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

Requirements of this nature are quite common, and can arise in many ways. One source of precedence constraints is physical structure. For example, a manufacturer must decide which items to produce in a fixed time period, but must produce the components for certain complex assemblies before producing the assemblies themselves.

Precedence constraints define a *partial order* "$\rightsquigarrow$" on the set of items; if item i must be selected whenever item j is selected, then $i \rightsquigarrow j$ (read i precedes j). Such a partial order can be represented by a directed graph $G(\rightsquigarrow) = (N, A)$; N contains a node for each item, and for each $i, j \in N$, A contains arc (i, j) if and only if $i \rightsquigarrow j$. The graph $G(\rightsquigarrow)$ is assumed to be acyclic, that is, there is no group of items, all of which must either be chosen or not chosen. The graph is also assumed to be connected. If this is not the case, consider the maximal connected components of $G(\rightsquigarrow)$; a "dummy" item can be added that precedes, or is preceded by, one item from each component.

Several kinds of partial orders are particularly interesting.

Definition 9 A partial order $\rightsquigarrow$ is a *tree partial order* if the undirected version of $G(\rightsquigarrow)$ is a tree. ■

Definition 10 A partial order $\rightsquigarrow$ is an *in-tree partial order* if it is a tree partial order and $G(\rightsquigarrow)$ contains some node (the "root") *toward* which all its arcs point. ■

Definition 11 A partial order $\rightsquigarrow$ is an *out-tree partial order* if it is a tree partial order and $G(\rightsquigarrow)$ contains some node (the "root") *away from* which all its arcs point. ■

Figure 8 shows an in-tree partial order, Figure 9 shows an out-tree partial order, and Figure 10 shows a general tree partial order that is neither an in-tree partial order nor an out-tree partial order. Figure 11 shows a partial order that is not a tree partial order.

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

Partially Ordered Knapsack Feasible Set

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

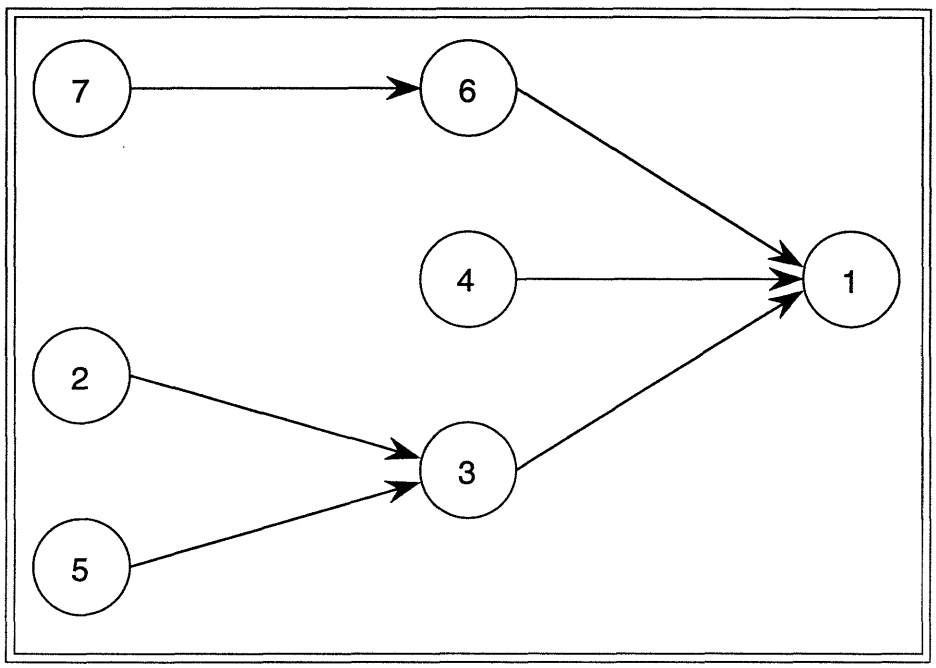

Figure 8: An in-tree partial order.

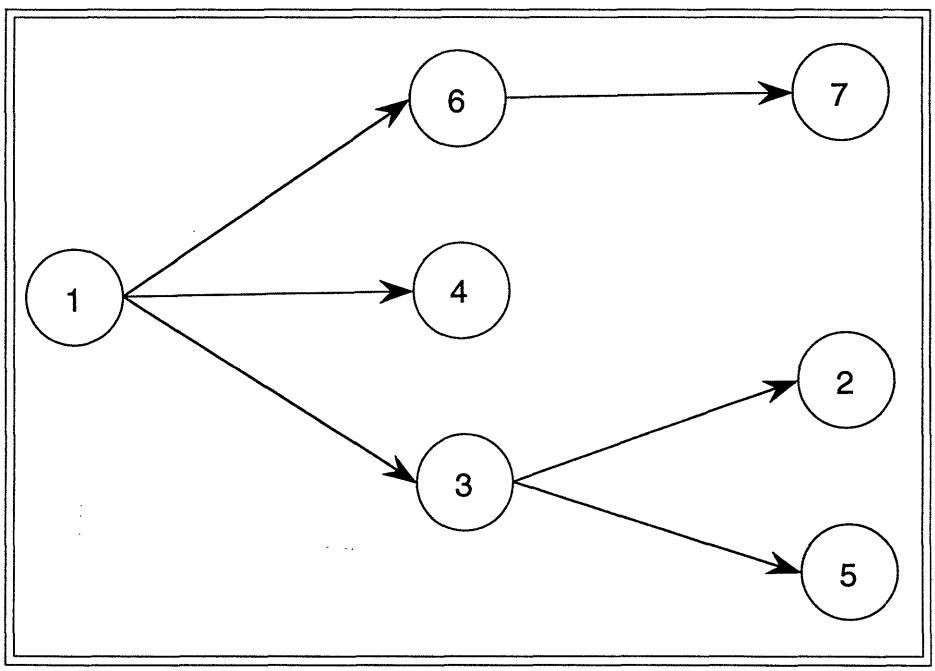

Figure 9: An out-tree partial order.

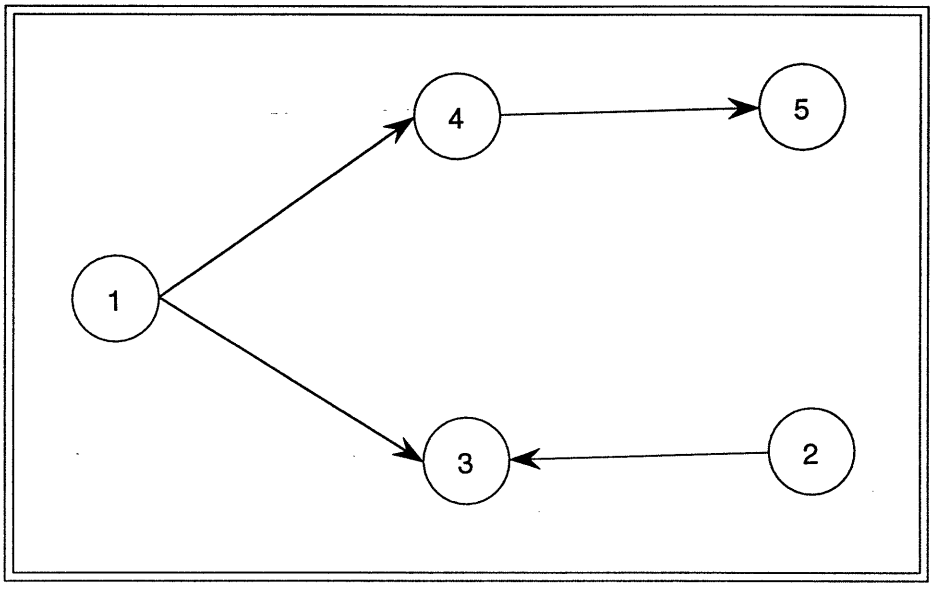

Figure 10: A general tree partial order.

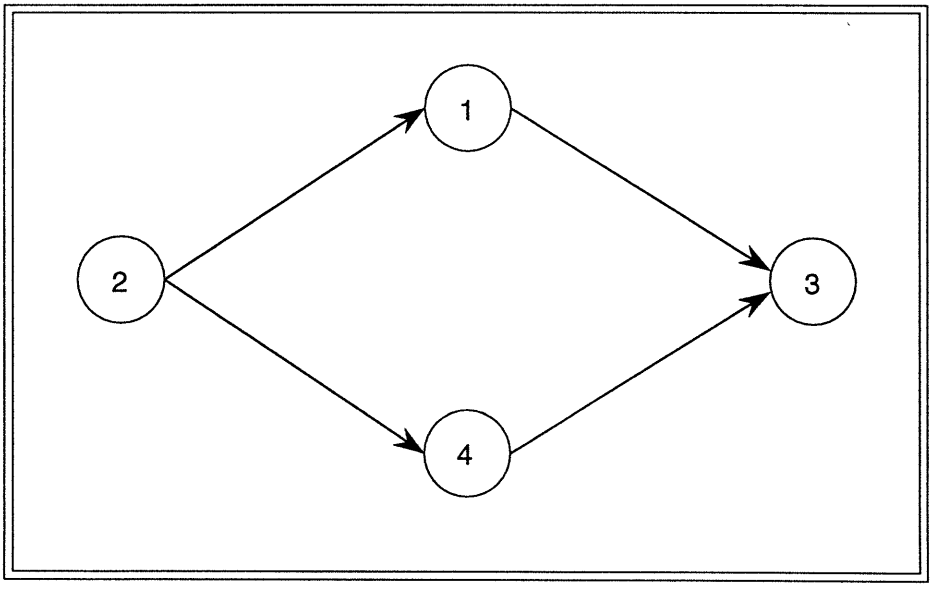

Figure 11: A partial order that is not a tree partial order.

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

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

The Partially Ordered Knapsack problem with a single criterion of the form $\sum_{j=1}^{n} p_j x_j$ is strongly $\mathcal{NP}$-hard when general partial orders are allowed, but can be solved in pseudo-polynomial time when only tree partial orders are allowed[GJ79]. A PTAS for the maximization form of this single-criterion problem, restricted to in-tree partial orders, is described in [IK78]; it runs in time $O\left(n^{2+1/\varepsilon}/\varepsilon\right)$. The algorithm of [IK78] is also applied to the minimization form of the problem in which the knapsack constraint is of the covering type and the precedence constraints form an in-tree partial order.

A variety of stronger results is presented in [JN83]. An FAS is presented for out-tree or in-tree partial orders. The FAS is based on the standard "bottom-up" dynamic programming approach and runs in time $O\left(n^3/\varepsilon^2 + n^3 \log(P^*)\right)$. A new dynamic programming paradigm, the "left-right" approach, that yields an FAS that runs in time $O\left(n^2/\varepsilon^2 + n^2 \log(P^*)\right)$ for out-tree partial orders and $O\left(n^3/\varepsilon\right)$ for in-tree partial orders is introduced in [JN83]. Here P^* is the optimal value, and the $\log(P^*)$ factors can be replaced by $\log(n)$ in the time bounds.

The following theorem generalizes the results of [JN83] by allowing multi-criteria objective functions.

Theorem 12 *Let Ψ be the collection of feasible regions for the Partially Ordered Knapsack problem with general tree partial orders and let $\omega \in \Omega^r$. Then the problem $(\Psi, \mathcal{AM}(\omega), \omega, Opt)$ has an FAS.*

Proof. The standard dynamic programming approach can be used to solve the STT Network Flow problem $\left(\Psi, \mathcal{AP}\left(\omega^{1,=}\right), \omega^{1,=}, \text{Opt}\right)$ in VPP time. An explanation of the details can be found, for example, in [JN83]. Applying Lemma 2 and Lemma 4 in [SO94] with Theorem 2 proves this theorem. ■

Reversing the constraints that define the feasible region produces a Partially Ordered Knapsack Covering problem with precedence constraints the *reverse* of those in the original problem. Application of Theorem 10 would yield a solution to the wrong problem.

Note, however, that if the original precedence constraints form a tree partial order, then the reversed constraints also form a tree partial order. If the original precedence constraints are reversed *before* the covering transformation is applied, then a covering problem with the original precedence constraints results. The following conclusion is therefore valid, even though not all the original constraints are reversed to pose the covering form of a Partially Ordered Knapsack problem.

Corollary 4 *The Partially Ordered Knapsack Covering problem with general tree partial orders has an FAS.*

9 Production Planning Problems

The production planning problem considered here is a capacitated single commodity dynamic lot sizing problem. The deterministic, dynamic demand must be satisfied, but may be backlogged. Time-dependent production capacities are specified. The object is to determine how many units, if any, of the item to produce each period in order to minimize the (possibly non-linear) costs of production and inventory storage over a finite horizon.

Given demands d_j, production capacities c_j, and production quantities x_j, define the following cumulative quantities:

$$\begin{aligned} D_j &= \sum_{i=1}^{j} d_i && \text{cumulative demand through period } j, \\ C_j &= \sum_{i=1}^{j} c_i && \text{cumulative capacity through period } j, \\ X_j &= \sum_{i=1}^{j} x_i && \text{cumulative production through period } j, \text{ and} \\ I_j &= X_j - D_j && \text{inventory at the end of period } j. \end{aligned}$$

The feasible set for this problem is defined as follows:

Capacitated Single Commodity Dynamic Lot Sizing Feasible Set

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

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

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

The assumption that $D_j \leq C_j$ is sufficient for the feasible set to be non-empty[FK71, FLRK80]. The constraint that $I_0 = 0$ can be removed, if desired; simply add another period $j = 0$ at the beginning in which $x_0 = I_0$ is produced at a suitably small cost[FLRK80]. The constraint that $I_n = 0$ is added without loss of generality[Zan66], and is commonly used to simplify calculations.

The cost functions typically used with this problem are:

$p_j(x_j)$ cost of producing x_j units in period j,
$h_j(I_j)$ cost of holding I_j units of inventory at the end of period j, and
$b_j(I_j)$ cost of backordering I_j units in period j.

This is an important problem that has received a lot of attention in the literature. A good review of previous results for this and related problems can be found in [BRG87]. A snapshot of the state-of-the-art with regard to known complexity results for variants of this problem appears in [FLRK80]. That paper has provided a framework for classifying many subsequent results.

The uncapacitated case, i.e. $c_j = \infty$, with concave functions p_j and h_j, was first treated in [WW58]. An $O\left(n^3\right)$ algorithm for the problem was presented in that paper. Backlogging for a limited number of periods is accommodated by an $O\left(n^2\right)$ algorithm that appears in [Zan66]. An $O\left(n^4\right)$ algorithm for the capacitated problem with constant capacities is described in [FK71]; time-dependent capacities are allowed in [LL82].

Linear production and holding costs and time-dependent set-up costs are handled by an algorithm in [BDMS78]; it runs in time $O\left(n^3 D_n + n D_n C_n\right)$. A modification that allows time-dependent capacities and backlogging and runs in time $O\left(n^4 D_n + n^2 D_n C_n\right)$ is described in [FLRK80]. More general production and holding costs are treated in [BY82]. The performance of various heuristics is studied in [BMY84, BM86], and an FAS for a continuous-time problem appears in [GJ90].

Note that quantities other than the production amount can be restricted. In [Lov73], for instance, a maximum and minimum inventory level for each period is specified. Also, most authors assume that the production cost functions are concave, most often consisting of a component consisting of a set-up cost and a component that is linear in the amount produced. An exception is [Vei64], in which convex cost functions are assumed.

The time bounds for previous algorithms for the problem considered here, although pseudo-polynomial, are not VPP. This means that none of them can be directly converted into an FAS in the manner described in [SO94]. The only exception is the FAS presented in [DO81].

Nevertheless, as the following theorem shows, the Capacitated Single Commodity Dynamic Lot Sizing problem does have an FAS. It therefore also has a VPP algorithm which, for some situations, may be preferable to the pseudo-polynomial algorithms already known.

Theorem 13 *Let Ψ be the collection of Capacitated Single Commodity Dynamic Lot Sizing feasible sets and $\omega \in \Omega$. Then $(\Psi, \mathcal{AM}(\omega), \omega, Opt)$ has an FAS.*

Proof. Let $\Pi_1 = (\Psi, \mathcal{AM}(\omega), \omega, \text{Feas})$ and $\Pi_2 = (\mathcal{C}_2, \mathcal{AM}(\omega), \omega, \text{Feas})$, where the feasible set $\mathcal{C}_2 = (A1/CG/M1/ND/FE)$. The reduction $\Pi_1 \propto_{\text{VPP}} \Pi_2$ will be shown. By Lemma 4 in [SO94]and Theorem 7, this will prove the present theorem.

Suppose that an instance $I_1 = (S, f, \omega, M)$ of Π_1 is given, with $S = S(n, d, c)$. The objective function f is

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

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

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

10 Summary

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

The computational complexity of finding approximate solutions to several problems that arise frequently in applications was also studied. The previously known FAS for the binary case of the single-criterion Arborescent Knapsack problem was generalized to the integer case with multiple criteria. An FAS for the covering form of this problem was also demonstrated; this problem has not been considered before except for the special case of the Min Job Sequencing with Deadlines problem.

An FAS for the multiple criteria form of the Partially Ordered Knapsack problem with general tree constraints was also shown, thereby generalizing the previously known single-criterion results. The covering form of this problem, which has not been discussed previously, was also shown to have an FAS.

Finally, the Capacitated Single Commodity Dynamic Lot Sizing problem was considered. The previously known pseudo-polynomial time algorithms for this problem were not VPP, and so could not be used directly to show the existence of an FAS. By reducing this problem to an STT Network Flow problem, however, the existence of an FAS was demonstrated. This also implies the existence of a VPP algorithm that might be more useful in some situations than are previously-known algorithms.

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

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

[BRG87] Harish C. Bahl, Larry P. Ritzman, and Jatinder N. D. Gupta. Determining lot sizes and resource requirements: A review. *Opns. Res.*, 35(3):329–345, May 1987.

[BDMS78] Kenneth R. Baker, Paul Dixon, Michael J. Magazine, and Edward A. Silver. An algorithm for the dynamic lot-size problem with time-varying production capacity constraints. *Management Sci.*, 24(16):1710–1720, December 1978.

[Bel61] Richard Bellman. *Adaptive Control Processes: A Guided Tour.* Princeton U. Press, Princeton, NJ, 1961.

[BY82] Gabriel R. Bitran and Horacio H. Yanasse. Computational complexity of the capacitated lot size problem. *Management Sci.*, 28(10):1174–1186, October 1982.

[BMY84] Gabriel R. Bitran, Thomas L. Magnanti, and Horacio H. Yanasse. Approximation methods for the uncapacitated dynamic lot size problem. *Management Sci.*, 30(9):1121–1140, September 1984.

[BM86] Gabriel R. Bitran and Hirofumi Matsuo. Approximation formulations for the capacitated lot size problem. *Opns. Res.*, 34(1):63–74, January 1986.

[DO81] M. Dada and James B. Orlin. The capacitated multi-item dynamic lot-size problem: Exact and ε-optimal algorithms. Presentation at Spring CORS/TIMS/ORSA Conference, Toronto, 1981.

[FK71] Michael Florian and Morton Klein. Deterministic production planning with concave costs and capacity constraints. *Management Sci.*, 18(1):12–20, September 1971.

[FLRK80] Michael Florian, Jan Karel Lenstra, and Alexander H. G. Rinnooy Kan. Deterministic production planning: Algorithms and complexity. *Management Sci.*, 26(7):669–679, July 1980.

[FF62] L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks.* Princeton U. Press, 1962.

[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.

[GJ90] Bezalel Gavish and Robert E. Johnson. A fully polynomial approximation scheme for single-product scheduling in a finite capacity facility. *Opns. Res.*, 38(1):70–83, January 1990.

[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.

[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.

[GLLRK79] Ronald L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. In Peter L. Hammer, Ellis L. Johnson, and Bernhard H. Korte, editors, *Discrete Optimization II*, pages 287–326. 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.

[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.

[Hen85] 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.

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

[JN83] David S. Johnson and K. A. Niemi. On knapsacks, partitions, and a new dynamic programming technique for trees. *Math. of Opns. Res.*, 8(1):1–14, February 1983.

[Knu73] Donald E. Knuth. *The Art of Computer Programming*, volume 1: Fundamental Algorithms. Addison-Wesley Publishing Co., Reading, MA, second edition, 1973.

[Kre77] David M. Kreps. Decision problems with expected utility criteria, I: Upper and lower convergent utility. *Math. of Opns. Res.*, 2(1):45–53, February 1977.

[LL82] Anne-Marie Lambert and Hanan Luss. Production planning with time-dependent capacity bounds. *Eur. J. Opnl. Res.*, 9:275–280, 1982.

[LM69] Eugene L. Lawler and J. M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Sci.*, 16(1):77–84, September 1969.

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

[LH90] Duan Li and Yacov Y. Haimes. New approach for nonseparable dynamic programming problems. *J. Opt. Theory and Appl.*, 64(2):311–330, February 1990.

[Lov73] Stephen F. Love. Bounded production and inventory models with piecewise concave costs. *Management Sci.*, 20(3):313–318, November 1973.

[RND77] Edward M. Reingold, Jurg Nievergelt, and Narsingh Deo. *Combinatorial Algorithms: Theory and Practice.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1977.

[Rot64] Michael H. Rothkopf. *Scheduling Independent Tasks on One or More Processors.* PhD thesis, School of Industrial Management, MIT, Cambridge, MA, January 1964.

[SO94] Hershel M. Safer and James B. Orlin. Fast approximation schemes for multi-criteria combinatorial optimization. October 1994.

[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.

[Sni87] Moshe Sniedovich. A class of nonseparable dynamic programming problems. *J. Opt. Theory and Appl.*, 52(1):111–121, January 1987.

[Vei64] Arthur F. Veinott, Jr. Production planning with convex costs: A parametric study. *Management Sci.*, 10(3):441–460, April 1964.

[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.

[WW58] Harvey M. Wagner and Thomson M. Whitin. Dynamic version of the economic lot size model. *Management Sci.*, 5(1):89–96, October 1958.

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

[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.

[Zan66] Willard I. Zangwill. A deterministic multi-period production scheduling model with backlogging. *Management Sci.*, 13(1):105–119, September 1966.