LA-UR 97-5198

CONF-980633--

TITLE: # MULTICRITERIA APPROXIMATION THROUGH DECOMPOSITION

AUTHOR(S): C. Burch, S. Krumke, M. Marathe, C. Phillips

SUBMITTED TO: Integer Programming & Combinatorial Optimization
Houston, Tx
June ~~August~~, 1998

Multicriteria approximation through decomposition

Carl Burch[*] Sven Krumke[†] Madhav Marathe[‡] Cynthia Phillips[§] Eric Sundberg[¶]

Abstract

We propose a general technique called *solution decomposition* to devise approximation algorithms with provable performance guarantees. The technique is applicable to a large class of combinatorial optimization problems that can be formulated as integer linear programs. Two key ingredients of our technique involve finding a decomposition of a fractional solution into a convex combination of feasible integral solutions and devising generic approximation algorithms based on calls to such decompositions as oracles. The technique is closely related to *randomized rounding*. Our method yields as corollaries unified solutions to a number of well studied problems and it provides the first approximation algorithms with provable guarantees for a number of new problems. The particular results obtained in this paper include the following:

1. We demonstrate how the technique can be used to provide more understanding of previous results and new algorithms for classical problems such as **Multicriteria Spanning Trees**, and **Suitcase Packing**.

2. We show how the ideas can be extended to apply to multicriteria optimization problems, in which we wish to minimize a certain objective function subject to one or more budget constraints. As corollaries we obtain first non-trivial multicriteria approximation algorithms for problems including the k-**Hurdle** and the **Network Inhibition** problems.

[*] 5000 Forbes Ave, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15217. E-mail: `cburch+@cmu.edu`. Some work completed while at Sandia National Laboratory. Supported in part by a National Science Foundation Graduate Fellowship.

[†] Department of Computer Science, University of Würzburg, Am Hubland, 97074 Würzburg, Germany. E-mail: `krumke@informatik.uni-wuerzburg.de`.

[‡] P O Box 1663, MS B265, Los Alamos National Laboratory, Los Alamos NM 87545. E-mail: `marathe@lanl.gov`. Work supported by the Department of Energy under Contract W-7405-ENG-36.

[§] Applied Mathematics Department, Sandia National Laboratory, P O Box 5800, Albuquerque, NM 87185. E-mail: `caphill@cs.sandia.gov`. Work supported in part by the United States Department of Energy under Contract DE-AC04-94AL85000.

[¶] Department of Computer Science, Rutgers University, NJ. E-mail: `sundberg@euler.rutgers.edu`. Some work completed while at Sandia National Laboratory.

1 Introduction

Many recent advances in approximation algorithms for combinatorial problems have formulated a problem as an integer program, solved the linear-programming relaxation of the problem, and then used the resulting fractional solution to guide the search for an integral feasible solution whose objective is close to the bound provided by the linear program. One successful approach is *randomized rounding*. We refer the reader to [MNR, Sri95, Sri97, ST97] for more discussion of related results. In this paper, we propose a *new* general method called *solution decomposition* for approximating the optimal value of certain types of linear integer programs (IP). Our technique applies to a wide variety of optimization problems, including a number of problems for which no previous (multicriteria) approximation results were known.

To simultaneously optimize two or more criteria, we optimize one criterion while enforcing a budget on all other criteria. That is, for a given set of solutions $\mathcal{S}$, and for given criteria $f_1, \ldots, f_k$ and budgets $B_1, \ldots, B_{k-1}$, we seek an element of $\mathcal{S}$ minimizing the criterion f_k and meeting the budgets:

$$\text{(GP)} \quad \text{minimize} \quad f_k(S)$$
$$\text{where} \quad \begin{cases} f_i(S) \le B_i & \forall\, 1 \le i \le k-1 \\ S \in \mathcal{S} \end{cases}$$

This is a *k-criteria problem* (due to the $k - 1$ budgeted criteria and the single objective criterion). Although in general the constraints need not be linear, in this paper we restrict our attention to problems for which the above is an *integer linear program* (IP). That is, the feasible region is the set of variables $x \in \mathcal{Z}^n$ such that $Ax \le b$ and $x \ge 0$, for $m \times n$ integer matrix A and m-vector b.

The strongest type of approximation bound for such problems is to guarantee a solution meeting the budgeted constraints and approximating the optimization ratio. Sometimes the achievable (not $\mathcal{NP}$-hard) bounds are disappointing, however; in these cases, it is worthwhile considering how to find a solution that approximates both the budgets and the objective function. This leads to the notion of *multicriteria approximation algorithms* [MRS+95]. Let S^* be the optimal solution meeting the budget constraints. An approximation algorithm $\mathcal{A}$ is called an $(\rho_1, \ldots, \rho_k, \beta)$*-approximation algorithm* if $\mathcal{A}$ outputs a solution $S \in \mathcal{S}$ such that $f_i(S) \le \rho_i B_i$ (it ρ_i-approximates each budgeted constraint f_i) and $g(S) \le \beta g(S^*)$ (β-approximates the objective function g).

1.1 Method overview

We now describe our method and give definitions needed for the remainder of the paper. The decomposition method partitions the task of designing a new approximation algorithm into three orthogonal issues: modeling the problem as a IP, devising decomposition algorithms, and designing approximation algorithms based on these decompositions.

Let IP be an optimization problem given by an integer program. The solution decomposition method has three steps. First we relax IP and solve the resulting linear program LP. Then we decompose the resulting fractional solution into a convex combination of feasible integral solutions. Finally, we choose one of the integral solutions (according to some criteria).

The second step of the general approach involves the construction of a *decomposition.* Let $\mathcal{S}$ represent the set of feasible solutions to a collection of integer constraints. For example, $\mathcal{S}$ could define the set of spanning trees. In this abstract we consider the integer program GP defined above where each f_i is restricted to be a *cost function*, namely a linear function with nonnegative coefficients over nonnegative variables. The constraint $f_i(S) \leq B_i$ is a *budget constraint*, and B_i is the *budget* for f_i.

A *decomposition* for a fractional solution $\tilde{S}$ (a rational setting for the variables) is a set $\{S^{(1)}, S^{(2)}, \ldots, S^{(N)}\} \subset \mathcal{S}$ with corresponding weights $\alpha^{(i)}$ so that $\sum_{i=1}^{N} \alpha^{(i)} = 1$, $\alpha^{(i)} \geq 0$, and, for each cost function f_j, $\sum_{i=1}^{N} \alpha^{(i)} f_j(S^{(i)}) \leq \rho_j f_j(\tilde{S})$ We say this decomposition ρ_j*-approximates* f_j. We can alternately view the set $\{S^{(1)}, S^{(2)}, \ldots, S^{(N)}\}$ with the associated coefficients $\alpha = \cup_i \alpha^{(i)}$ as a probability space; each $S^{(i)}$ denotes an event and $\alpha^{(i)}$ denotes the probability that the event will occur. We will sometimes call this the α*-distribution*.

A *decomposition algorithm* finds such a decomposition for any $\tilde{S}$ satisfying the relaxed LP constraints. In general, the number of solutions N can be quite large, and so a decomposition algorithm, if it were to list each $S^{(i)}$, can take exponential time. We restrict our study to two forms of decomposition algorithms useful for polynomial-time approximations. In a *deterministic decomposition algorithm (DDA)*, N is polynomial in the input size, and all N solutions are computed in polynomial time. A *randomized decomposition algorithm (RDA)* samples from the (potentially exponentially large) solution space according to the α distribution in polynomial time. In other words, for a given $\tilde{S}$ satisfying the relaxed constraints for $\mathcal{S}$, the RDA takes polynomial time to produce a solution $S \in \mathcal{S}$ such that $\mathrm{E}\left[f_j(S)\right] \leq \rho_j f_j(\tilde{S})$ for each f_j.

Decompositions that satisfy the inequality $\sum_{i=1}^{N} \alpha^{(i)} x^{(i)} \leq \rho \tilde{x}$ for *every variable* x in an IP are ρ*-approximate decompositions*. We sometimes refer to 1-approximate decompositions as *exact decompositions*. These decompositions can be used with any cost function.

1.2 Applications

We discuss some applications of our techniques, defining those that are less well known.

In the **Network Inhibition** problem, we are given a graph $G(V, E)$ with two designated vertices $s, t \in V$, and each edge has an initial capacity $c_{u,v}$ and a removal cost $r_{u,v}$ representing the cost to remove the edge from the graph. We wish to expend up to a fixed budget B on edge removals to minimize the maximum s-t flow in G. Destruction is linear, so that paying $\alpha r_{u,v}$ for $0 \leq \alpha \leq 1$ removes $\alpha c_{u,v}$ units of capacity from edge (u, v). Phillips first characterized the complexity of this problem [Phi93]. She showed the problem is strongly $\mathcal{NP}$-complete for general graphs, but gave no approximation algorithms for the general case. In this paper we give a simple decomposition algorithm; using this result in conjunction with our the general result in Section 2 gives a $(1+\epsilon, 1+1/\epsilon)$-approximation algorithm.[1] Rao, Shmoys, and Tardos have independently achieved similar results for the single-budget case using a parametric-search approach [Shmon].

In the k-**Hurdle** problem we are given an undirected graph $G(V, E)$ and two nodes $s, t \in V$. We wish to find a minimum-cost set of edges $E' \subset E$ so that every path in (V, E) from s to t crosses at least k edges in E'. The **1-Hurdle** problem is the **Minimum** s-t **Cut** problem. The more general question arises in physical security applications, where one would like to place security cameras in a building so that a thief must pass at least k cameras between the entrance point and a goal site. To our knowledge, the k-**Hurdle** problem has not been previously studied. The DDA we present gives a simple polynomial-time algorithm and implies extensions to multicost versions.

In Section 3, we also consider decomposition-based approximations for multicriteria versions of the **Suitcase-Packing** problem (the complement of the knapsack problem), set cover, separating k pairs of vertices, and the **Shortest-Paths** problem.

1.3 Comparison to related work

The work presented here is closely related to two techniques used in the past to approximate single-criterion and multicriteria optimization problems: randomized rounding, and parametric search.

In *randomized rounding*, one typically formulates a given optimization problem by an integer program, relaxes the integrality constraints, solves the resulting linear program, and finally rounds the fractional solutions to obtain feasible integral solutions [RT87, MNR]. Algorithms based on solution decompositions are similar in spirit to algorithms based on randomized rounding; but they differ crucially in the formalization. The decomposition formalization allow the development of multicriteria algorithms using the decomposition algorithm as an oracle.

Parametric search, proposed in [KNR$^+$96, MRS$^+$95] can be applied to multicriteria problems where all criteria are similar. For example, one can apply parametric search to find a Steiner tree of minimum diameter under one metric among all trees whose diameter under another metric meets a given bound. The method requires an approximation algorithm for solving the single-criterion problem. In parametric search, a multicriteria optimization problem is reduced to a single-criteria problem using a weighted combination of the criteria.

A result similar to Theorem 3 applies for parametric search [MRS$^+$95]. (Although bound (1) has not been claimed earlier, it is easy to derive from published proofs.) This result applies regardless of the linearity of constraints, but the constraints must be on the for similar objectives [MRS$^+$95]. It can also be applied to problems where one has a budget for changing the coefficients of the objective [KNR$^+$96]).

Algorithms based on solution decomposition techniques yield similar bounds for a number of the problems studied with

[1]The result is actually stronger, since we can guarantee either a feasible solution or a superoptimal solution; see Section 3 for more details.

parametric search. It also applies to several new problems where the criteria are dissimilar. Furthermore, decomposition-based methods scale polynomially with more criteria. We elaborate more on this in Section 2.

The decomposition algorithms are similar to two previous papers. Naor and Schieber consider using the *Edmonds decomposition* (developed in [Edm73, Eve79, GM95]) for s-branchings (rooted spanning trees) to get approximation algorithms for spanning trees [NS97]. Although there is an error in their specific application (**Bounded-Diameter Minimum Spanning Trees**), the approach is still useful. Significantly, the techniques also apply to RDAs, opening the possibility of using randomized rounding techniques. Some other recent work by Srinivasan and Teo give constant factor approximation algorithms for certain global packet routing problems [ST97]. Although they do not consider the general framework considered in this paper, Theorem 4 in their paper is very similar in spirit to the work done here.

The remainder of the paper is organized as follows: Section 2 describes how decompositions can be used to obtain multi-criteria approximations for integer programs with 0, 1, 2, or more side budget constraints. In Section 3 we give new decompositions for the k-**Hurdle** problem, **Network Inhibition** problem, and s-t cuts. We use these new decompositions to find polynomial-time algorithms for the k-**Hurdle** problem, bicriteria approximations for **Network Inhibition**, and approximations for multi-cost cut problems.

2 Multicriteria approximations via decomposition

In this section we present general general algorithms that construct approximation algorithms for (GP) using decomposition algorithms as an oracle. We first show how to accomplish this for programs with no budget constraints, yielding true approximation algorithms. We then give an algorithm for programs with one or more budget constraints. Finally, we consider the case when the number of budget constraints grows with the problem size.

First consider the case of a single criterion. That is, we wish to minimize $f(S)$ such that $S \in \mathcal{S}$. The following theorems (proof omitted) shows that these decompositions are powerful.

Theorem 1 *If there is a DDA for $\mathcal{S}$ that ρ-approximates a cost function f, then there is also a ρ-approximation algorithm for the problem of minimizing f on $\mathcal{S}$.*

Theorem 2 *If there is a RDA for $\mathcal{S}$ that ρ-approximates a cost function f, then for any $\epsilon > 0$ there is a $(\rho + \epsilon)$-approximation algorithm to minimize f on $\mathcal{S}$ The expected number of draws from the RDA is at most $1 + (\rho - 1)/\epsilon$. If $\rho = 1$, we can find an optimal solution with a single draw from the RDA.*

The one-criterion results are simple, but they are useful for determining what we can reasonably expect from a decomposition algorithm for a given $\mathcal{S}$. For example, finding an exact decomposition algorithm (deterministic or randomized) for Steiner trees would imply $\mathcal{P} = \mathcal{NP}$.

2.1 Two criteria

In a two-criteria problem, we have a single budget constraint.

$$\text{minimize} \quad f_2(S)$$
$$\text{where} \quad \begin{cases} f_1(S) \leq B \\ S \in \mathcal{S} \end{cases}$$

Although we do not guarantee that the decomposition can meet the budget, we can guarantee that it does not exceed its budget dramatically as shown in the following theorems. Note that the solution is either a $(1, 1 + 1/\gamma)$-approximation or a $(1 + \gamma, 1)$-approximation, but we do not know *a priori* which we will get. The parameter γ biases this choice. As the budget is violated more (up to the maximum $1 + \gamma$ factor), the residual capacity is moving linearly toward zero.

Theorem 3 *Given a DDA for $\mathcal{S}$ that ρ_2-approximates f_2 and ρ_1-approximates f_1, for any $\gamma > 0$ in polynomial time we can find a solution S such that*

$$\frac{f_1(S)}{\rho_1 B} + \gamma \frac{f_2(S)}{\rho_2 f_2(S^*)} \leq 1 + \gamma \tag{1}$$

where S^* is the integer program's optimal solution.

Proof. Let $\tilde{S}$ be the optimal, fractional solution to the relaxed linear program. We show that some solution S given by the DDA satisfies

$$\frac{f_1(S)}{\rho_1 B} + \gamma \frac{f_2(S)}{\rho_2 f_2(\tilde{S})} \leq 1 + \gamma\,. \tag{2}$$

Since $f_2(S^*) \geq f_2(\tilde{S})$, this solution will satisfy (1).

Consider a random S drawn from the α distribution. We expect it to satisfy (2):

$$\begin{aligned}
&\mathrm{E}\left[\frac{f_1(S)}{\rho_1 B} + \gamma \frac{f_2(S)}{\rho_2 f_2(\tilde{S})}\right] \\
&\quad = \frac{\mathrm{E}[f_1(S)]}{\rho_1 B} + \gamma \frac{\mathrm{E}[f_2(S)]}{\rho_2 f_2(\tilde{S})} \\
&\quad \leq \frac{\rho_1 f_1(\tilde{S})}{\rho_1 B} + \gamma \frac{\rho_2 f_2(\tilde{S})}{\rho_2 f_2(\tilde{S})} \\
&\quad \leq 1 + \gamma
\end{aligned}$$

Hence one of the DDA's solutions satisfies (2). ■

Theorem 4 *Given an RDA for $\mathcal{S}$ that ρ_2-approximates f_2 and ρ_1-approximates f_1, for any $\gamma > 0$ and $\epsilon > 0$, we can find a solution S such that*

$$\frac{f_1(S)}{\rho_1 B} + \gamma \frac{f_2(S)}{\rho_2 f_2(S^*)} \leq (1 + \epsilon)(1 + \gamma)\,,$$

where S^* is the integer program's optimal solution. We expect at most $1 + 1/\epsilon$ draws from the RDA.

Proof. Let $\tilde{S}$ be the relaxed program's optimal solution. The algorithm is to continue drawing until we find a solution S such that

$$\frac{f_1(S)}{\rho_1 B} + \gamma \frac{f_2(S)}{\rho_2 f_2(\tilde{S})} \leq (1+\epsilon)(1+\gamma)$$

Since $f_2(S^*) > f_2(\tilde{S})$, outputting this solution will give the desired bound.

Consider a random S drawn from the distribution, and let X denote the value of

$$\frac{f_1(S)}{\rho_1 B} + \gamma \frac{f_2(S)}{\rho_2 f_2(\tilde{S})}$$

As in the proof of Theorem 3, the expected value of X is at most $1+\gamma$. We employ Markov's inequality to bound the chance that a given draw is bad.

$$\begin{aligned} \Pr[X > (1+\epsilon)(1+\gamma)] &\leq \Pr[X > (1+\epsilon)\mathrm{E}[X]] \\ &\leq \frac{1}{1+\epsilon} \end{aligned}$$

On any draw we terminate with probability at least $\epsilon/(1+\epsilon)$; hence we expect to take $1 + 1/\epsilon$ draws to terminate. ■

A geometric view: Although the proofs for Theorems 3 and 4 are probabilistic, we can also interpret the results geometrically. The geometric view motivates the choice of γ and explains why one should not expect better bounds from decomposition algorithms, at least if applied only to the relaxed program's solution. We can view any solution S in $\mathcal{S}$ as a point at $(f_1(S), f_2(S))$ in the two-dimensional plane.

Consider the convex hull of the points for $\mathcal{S}$. Because any solution to the LP relaxation of the IP can be expressed as a convex combination of integer solutions via the DDA, the set of feasible solutions to the LP relaxation is contained in this convex hull. Vertices of the lower convex hull correspond to budget values for f_1 where the LP solution is integral. For other budget values, the LP optimum is on a segment of the convex hull defined by two integral solutions: one that meets the budget, and another that exceeds the budget but is superoptimal. An exact DDA gives only solutions on this convex hull segment. Consider the line ℓ given by (2) with $\rho_1 = \rho_2 = 1$. Parameter γ controls the slope of ℓ. Line ℓ passes through point $p = (B, f_2(\tilde{S}))$, where $\tilde{S}$ is the LP optimum, and therefore it intersects the convex hull segment defined by the DDA solutions. Since the DDA gives points (solutions) on either side of point p, at least one of these must lie below the line, and this is the solution output.

2.2 Many criteria

The above theorems generalize to multiple criteria. Geometrically, we take the solution to a high-dimensional space. Consider the integer program GP with which the paper began.

Theorem 5 *Given a decomposition algorithm for $\mathcal{S}$ that ρ_i-approximates each f_i, and given weights γ_i such that $\sum_i 1/\gamma_i = 1$, and $\epsilon > 0$, we can find a solution S such that*

$$\sum_{i=1}^{k-1} \frac{f_i(S)}{\gamma_i \rho_i B_i} + \frac{f_k(S)}{\gamma_k \rho_k f_k(S^*)} \leq 1 + \epsilon,$$

where S^* is the integer program's optimal solution. If we have a DDA, we can take $\epsilon = 0$. If we have an RDA, we expect to use $1 + 1/\epsilon$ draws from the RDA.

2.3 More than many criteria

Sometimes the number of constraints k in GP grows with the problem size; in this case, the linear bound of Theorem 5 is unacceptable. Our algorithm for this case requires a concept of *union* between solutions. Specifically, given $\{S^1, \ldots, S^N\} \subset \mathcal{S}$, we must be able to find $\cup_j S^j \in \mathcal{S}$ so that $g(\cup_j S^j) \leq \sum_j g(S^j)$ and, for each i, $f_i(\cup_j S^j) \leq \min_j f_i(S^j)$. This notion of union is very powerful. Finding similar bounds for weaker notions of union is an open problem.

Theorem 6 *Let S^* be the optimal solution to GP, and say we have a decomposition algorithm for $\mathcal{S}$ that 1-approximates g and each f_i. Given a concept of union, for any $\gamma > 0, \epsilon > 0$, we can find a solution $S \in \mathcal{S}$ so that*

$$g(S) \leq (2+\gamma)\frac{\ln k}{\ln \frac{1+\gamma}{\gamma+\epsilon}} g(S^*)$$

and, for all $1 \leq i \leq k$,

$$f_i(S) \leq \left(1 + \frac{2}{\gamma}\right) B_i \, .$$

If we have a DDA, we can take $\epsilon = 0$. If we have an RDA, we expect to use $O(\frac{1+\gamma^2}{\epsilon} \log k)$ draws.

In the DDA case, the proof for Theorem 6 is very similar to that of Naor and Schieber [NS97]. We prove the RDA case in the appendix.

3 Applications

3.1 The k-hurdle problem

In this section, we give an exact DDA for the k-**Hurdle** problem defined in Section 1 and some generalizations. The set of edges selected for hurdles is a form of cut, so we will sometimes refer to a solution as a k-hurdle cut. This DDA gives a polynomial-time algorithm, which can be more easily recognized by noticing that the matrix for the integer program is totally unimodular.[2] However, this DDA immediately gives approximation results for the multicriteria k-hurdle problem, and it will be used in later sections.

[2]A matrix $\mathcal{A}$ is *totally unimodular* if each square submatrix of $\mathcal{A}$ has determinant $0, +1$ or -1. That an integer program with totally unimodular constraint matrix can be solved in polynomial time is well-known (see [AMO93]).

In the integer program $\mathcal{H}$ for the k-**Hurdle** problem, variable $z_{u,v}$ is 1 if edge (u, v) contains a hurdle and 0 otherwise. Variable d_v represents the minimum number of hurdles on any path from s to v.

$$(\mathcal{H}) \quad \text{minimize} \quad \sum_{(u,v)\in E} c_{uv} z_{uv}$$

$$\text{where} \quad \begin{cases} d_v \leq d_u + z_{u,v} & \forall (u,v) \in E \\ d_u \leq d_v + z_{u,v} & \forall (u,v) \in E \\ d_s = 0, \ d_t = k & \\ z_{u,v} \in \{0,1\} & \forall (u,v) \in E \\ d_v \in \{0,1,\ldots,k\} & \forall v \in V \end{cases}$$

This is exactly the formulation for the **Minimum Cut** problem, except that we require k cuts between s and t. Since it differs from the integer program for the **Minimum Cut** problem only in its right-hand side, it is totally unimodular, so we can find an integer solution in polynomial time. Furthermore, generalizing to the case where some hurdles are already in place on the edges and/or multiple hurdles can be added to an edge (at constant unit cost) still leaves the constraint matrix totally unimodular, implying that these problems are polynomial-time solvable.

Nevertheless, the DDA for the k-hurdle problem is a building block for other decompositions and, through Theorems 3 and 5, it gives approximations for the generalization where there are secondary costs associated with placing hurdles on edges. For example, the set of hurdles could represent heterogeneous security options (cameras, guards, etc) where there are tradeoffs between initial cost, maintenance costs, delays to legitimate users, etc.

Theorem 7 *There is an exact DDA for the k-hurdle problem which decomposes a fractional solution to the linear-programming relaxation of IP $\mathcal{H}$ into at most $n-1$ integer feasible solutions.*

The proof of the decomposition is in the appendix. Given the above decomposition and Theorem 5, bounds for the **multicost cut problem** immediately follow. Given multiple costs on each edge, we wish to minimize one cost while obeying a budget constraint on each other cost. We can also add constraints bounding from above the total weight of nodes at least ℓ hurdles away from the source.

3.2 Network inhibition

In this section we show how the s-t cut decomposition given in Section 3.1 can be used to get the first approximation algorithms for the network inhibition problem, defined in Section 1.2.

Phillips [Phi93] observes that for a particular cut the greedy attack strategy is optimal. One removes edges in decreasing order of $c_{u,v}/r_{u,v}$ until the budget is depleted. Thus a solution to the network inhibition problem can be expressed as an s-t cut, which is then attacked in this manner.

In the integer-programming formulation for the network inhibition problem, variables d_v represent the cut. That is, $d_v = 1$ if vertex v is on the t side of the cut and $d_v = 0$ otherwise. Variable $z_{u,v}$ is the fraction of (u, v) removed through payment, and x_{ij} is the fraction remaining, which contributes to the residual capacity if edge (u, v) is in the cut.

If an edge (u, v) is in this cut (d_u and d_v differ) then we have $x_{u,v} + z_{u,v} = 1$. Thus we must pay to remove the edge (first constraint), or pay for the remaining capacity in the objective function. This integer program is related to IP $\mathcal{H}$ from Section 3.1, but, clearly (from the first constraint), it is not totally unimodular. However, the decomposition given in Section 3.1 is still useful. The decomposition and proof are in the appendix.

Theorem 8 *There is a DDA for the network-inhibition integer program relaxation which expresses the fractional solution as a convex combination of at most $n - 1$ attack strategies that together exactly approximate the budget constraint and objective function (residual capacity).*

3.3 Randomized rounding and Suitcase Packing

Many approximation algorithms work by *randomized rounding* (surveyed in [RT87, MNR]). The concept of an RDA is actually a formalization of randomized rounding, at least in its more elementary forms. Hence many established randomized rounding algorithms yield RDAs for interesting constraint sets. These include a $O(\log n)$-approximate RDA for **set cover** and an RDA for k**-pair cuts** that $O(\log k)$-approximates any capacity function [BV94].

As a simple example of using randomized rounding to get an RDA, we consider the **Suitcase Packing** problem. **Suitcase Packing**, like the **Knapsack** problem asks how to pack a bag so that the total weight does not exceed a budget W, but, rather than maximize the value of the contents, it asks to minimize the value of what is left. (We invert the problem because the techniques apply only when cost functions are to be minimized.) Say we have n_a instances of item a, each having weight w_a and badness c_a. For each item type the integer program formulation has a integer variable x_a indicating how many a's to pack and a variable y_a telling how many to leave behind. Then our integer program is the following.

$$\text{(SP)} \quad \text{minimize} \quad \sum_a c_a y_a$$
$$\text{where} \quad \begin{cases} \sum_a w_a x_a \leq B & \\ x_a \in \{0, 1, \ldots, n_a\} & \forall a \\ y_a = n_a - x_a & \forall a \end{cases}$$

Randomized rounding yields the following RDA.

Theorem 9 *We have an exact RDA for suitcase packing. It can be queried in $O(m)$ time.*

Proof. Given a solution of $\tilde{x}_a$, the RDA takes x_a to be $\lfloor \tilde{x}_a \rfloor + 1$ with probability $\tilde{x}_a - \lfloor \tilde{x}_a \rfloor$ and $\lfloor \tilde{x}_a \rfloor$ otherwise. The expected value of $\tilde{x}_a$, then, is x_a. ■

(In fact, for suitcase packing there is a simple DDA. Choose $y \in [0,1]$, and let x_a be $\lfloor \tilde{x}_a \rfloor$ if $\lfloor \tilde{x}_a \rfloor + y > \tilde{x}_a$; otherwise, choose $\lfloor \tilde{x}_a \rfloor + 1$. This is a randomized construction, but note that there are only $n+1$ choices for y resulting in different packings. We can easily list all of these.)

Although one may beat the bounds this technique implies for **Suitcase Packing** using more sophisticated techniques, this example illustrates the simplicity of using decompositions, which extend easily to multiple criteria. For example, we could add constraints on volume, monetary value, or a spouse's preferences. By using Theorem 5 we can find a reasonable solution approximating all these bounds.

3.4 s-t paths

Exact decompositions also exist for paths between nodes s and t in a graph. Using a flow formulation we can write linear constraints Q describing an s-t path.

$$\begin{array}{ll}
\sum_{v \in V} x_{v,t} = 1 & \\
\sum_{v \in V} x_{s,v} = 1 & \\
\sum_{u \in V} x_{u,v} = \sum_{u \in V} x_{v,u} & \forall v \neq s,t \\
x_{u,v} \in \{0,1\} & \forall u,v \in V
\end{array}$$

Variable $x_{u,v} = 1$ if and only if edge (u,v) is on the path and u precedes v. The first three constraints require a single unit of flow to be routed from s to t. Since the "flows" are constrained to be integral by the final constraint, we will get a single path from s to t.

Theorem 10 *We have an exact DDA for the above set of linear constraints.*

Proof. Use the path filtering procedure in [ST97] or the more efficient procedure in the appendix. ■

One can find a shortest path in polynomial time and there is a FPTAS for bicriteria shortest paths (see, e.g. [Phi93] for an efficient scheme and a discussion of previous work). We can use this decomposition to give multicriteria approximations for a bounded number of cost functions. There are instances where the result can be as bad as an $O(k)$-approximation relative to the LP bound for k criteria. However, this seems to be primarily related to weakness of the LP bound.

The authors would like to acknowledge helpful discussions with Avrim Blum, Adam Kalai, Bruce Maggs, and Santosh Vempala.

References

[AMO93] R Ahuja, T Magnanti, and J Orlin. *Network Flows*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1993.

[BV94] D Bertsimas and R Vohra. Linear programming relaxations, approximation algorithms, and randomization: a unified view of covering problems. Technical Report OR 285-94, MIT, 1994.

[Edm73] J Edmonds. Edge-disjoint branchings. In *Combinatorial Algorithms*, pages 91–96, 1973.

[Eve79] S Even. *Graph Algorithms*. Computer Science Press, 1979.

[GM95] H Gabow and K Manu. Packing algorithms for arborescences (and spanning tre es) in capacitated graphs. In *4th MPS Conferences on Integer Programming and Combinatorial Optimization*, pages 388–402, 1995.

[Hoc95] D S Hochbaum. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, Boston, MA, 1995.

[KNR+96] S Krumke, H Noltemeier, S Ravi, M Marathe, and K Drangmeister. Modifying networks to obtain low cost trees. In *22nd International Workshop on Graph-Theoretic Concepts in Computer Science, Cadenabbia, Italy*, volume 1197 of *Lecture Notes in Computer Science*, pages 293–307, June 1996.

[MNR] R Motwani, J Naor, and P Raghavan. Randomized approximation algorithms in combinatorial optimization. In [Hoc95].

[MRS+95] M Marathe, R Ravi, R Sundaram, S Ravi, D Rosenkrantz, and H Hunt III. Bicriteria network design problems. In *22nd International Colloquium on Automata, Languages, and Programming To appear in J. Algorithms*, pages 487–98, 1995.

[NS97] J Naor and B Schieber. Improved approximations for shallow-light spanning tre es. In *38th IEEE Symposium on Foundations of Computer Science*, pages 536–541, 1997.

[Phi93] C Phillips. The network inhibition problem. In *25th ACM Symposium on the Theory of Computing*, pages 288–93, 1993.

[RT87] P Raghavan and C Thompson. Randomized rounding: a technique for provably good algorithms. *Combinatorica*, 7:365–74, 1987.

[Shmon] D Shmoys, Personal communication.

[Sri95] A Srinivasan. Improved approximations of packing and covering problem. In *ACM Symposium on the Theory of Computing*, pages 268–276, 1995.

[Sri97] A Srinivasan. Improved approximations for edge disjoint paths, unsplittable flows and related routing problems. In *IEEE Foundations on Computer Science*, 1997.

[ST97] A Srinivasan and C.-P. Teo. A constant-factor approximation algorithm for packet routing, and balancing local vs global criteria. In *ACM Symposium on the Theory of Computing*, pages 636–643, 1997.

Appendix A: Using RDAs for many-criteria problems

Proof of Theorem 6. Our algorithm is the following. Find the solution $\tilde{S}$ to the relaxed program. Begin with $F = \{1, \ldots, k\}$. We proceed in phases. In the jth phase we continue drawing from the RDA until we find a solution S^j so that $g(S^j) \leq (2+\gamma)g(\tilde{S})$ and for at least a $(1-\epsilon)/(1+\gamma)$ fraction of $i \in F$ we have $f_i(S^j) \leq (1+2/\gamma)B_i$. We remove these i from F and continue to the next phase until $F = \emptyset$. Since each phase decreases the size of F by a factor of at least $(\gamma+\epsilon)/(1+\gamma)$, by the end we have at most $\log_{(1+\gamma)/(\gamma+\epsilon)} k$ solutions. We return their union, S.

Since for each $1 \leq i \leq k$ we know that one of the solutions S^j has $f_i(S^j) \leq (1+2/\gamma)B_i$, we know that their union also estimates this constraint. Also, we know that, for each S^j, $g(S^j) \leq (2+\gamma)g(\tilde{S})$; so $g(\cup_j S^j)$ is bounded by

$$g(\cup_j S^j) \leq (2+\gamma)\log_{\frac{1+\gamma}{\gamma+\epsilon}} k g(\tilde{S}) \leq (2+\gamma)\frac{\ln k}{\ln\frac{1+\gamma}{\gamma+\epsilon}} g(S^*) .$$

It remains to show that the algorithm terminates. Call a solution S *eligible* if $g(S) \leq (2+\gamma)g(\tilde{S})$. We first see how often the RDA finds eligible solutions. We know that $\mathrm{E}[g(S)] \leq g(\tilde{S})$, so Markov's inequality bounds the chance a draw is ineligible.

$$\Pr\left[g(S) > (2+\gamma)g(\tilde{S})\right] \leq \frac{1}{2+\gamma}$$

So we expect to find an eligible S at least every $(2+\gamma)/(1+\gamma)$ draws.

Now what is the probability that a given f_i is $(1+2/\gamma)$-approximated by this S? By our previous reasoning, at least $(1+\gamma)/(2+\gamma)$ of the draws are eligible. Because f_i is always positive, we know that

$$\mathrm{E}\left[f_i(S) | g(S) \leq (2+\gamma)g(\tilde{S})\right] \leq \frac{2+\gamma}{1+\gamma}B_i .$$

(If it were more, $\mathrm{E}[f_i(S)]$ would be more than B_i.) This allows us to use Markov's inequality once again.

$$\Pr\left[f_i(S) > \frac{2+\gamma}{\gamma}B_i | g(S) \leq (2+\gamma)g(\tilde{S})\right] \leq \frac{\gamma}{1+\gamma}$$

Now we compute the probability that S is not selected. For the ith value in F, let X_i be a random variable that will be 1 if $f_i(S) > (1+2/\gamma)B_i$. The expected value of each X_i is $\gamma/(1+\gamma)$, so the expected value of $\sum_i X_i$ is $(\gamma/(1+\gamma))|F|$. We employ Markov's inequality one last time to get the probability S is not selected.

$$\Pr\left[\sum_i X_i > \left(1 - \frac{1-\epsilon}{1+\gamma}\right)|F|\right] \leq \frac{\gamma}{\gamma+\epsilon}$$

So we expect the phase to end after selecting $(\gamma+\epsilon)/\epsilon$ eligible solutions from the RDA.

Hence we expect to terminate in

$$\left(\frac{2+\gamma}{1+\gamma}\right)\left(\frac{\gamma+\epsilon}{\epsilon}\right)\left(\frac{\ln k}{\ln\frac{1+\gamma}{\gamma+\epsilon}}\right) = O\left(\frac{1+\gamma^2}{\epsilon}\ln k\right) .$$

draws from the RDA. ■

Appendix B: k-hurdle decomposition

Proof of Theorem 7. Let $\tilde{d}_v$ be the value of variable d_v in an optimal solution to LP relaxation for IP $\mathcal{H}$, and let $f_v = \tilde{d}_v - \lfloor \tilde{d}_v \rfloor$ be the fractional portion of $\tilde{d}_v$ for $v = 1, \ldots, n-2$. Define $f_s = 0$ and $f_t = 1$. Sort the f_v and call the resulting list $0 = x_0 \leq x_1, \leq \cdots \leq x_{n-2} \leq x_{n-1} = 1$. For each x_i we create a k-hurdle cut of weight $\alpha^i = x_{i+1} - x_i$ by setting d^i_u to $\lfloor \tilde{d}_u \rfloor$ if $x_i \geq f_u$ and $\lfloor \tilde{d}_u \rfloor + 1$ otherwise, and setting $z^i_{u,v} = |d^i_u - d^i_v|$. It is easier to think about the RDA: to sample, choose $x \in [0,1]$ uniformly at random. We take d_v to be $\lfloor \tilde{d}_v \rfloor$ if $x \geq f_v$ and $\lfloor \tilde{d}_v \rfloor + 1$ otherwise. Intuitively, the nodes are divided into k groups based upon the integer portion of $\tilde{d}_v$ (number of whole hurdles in front of v). Initially, every node in the group is on the "t side", and hurdles are placed on incoming edge. One by one, nodes move to the s side, so that as the value of x_i increases, a "wave" passes through each class.

Now we verify that $\tilde{d}_v$ is a convex combination of the d^i_v. Let $x_j = f_v$ (j is the index of f_v in the sorted list). Then we have

$$\begin{aligned}
\sum \alpha^i d^i_v &= (\lfloor \tilde{d}_v \rfloor + 1) \sum_{i=0}^{j} (x_i - x_{i-1}) + \lfloor \tilde{d}_v \rfloor \sum_{i=j+1}^{n-1} (x_i - x_{i-1}) \\
&= \lfloor \tilde{d}_v \rfloor + \sum_{i=1}^{j} (x_i - x_{i-1}) \\
&= \lfloor \tilde{d}_v \rfloor + x_j = \tilde{d}_v
\end{aligned}$$

We now verify that the LP values for the $z_{u,v}$ variables are also exactly decomposed. That is $\tilde{z}_{u,v} = \sum_i \alpha^i z^i_{u,v}$. The constraints demand that $z_{u,v}$ be at least $|d_u - d_v|$, and the objective function demands that $z_{u,v}$ be as small as possible, so we have $\tilde{z}_{u,v} = |\tilde{d}_u - \tilde{d}_v|$. Suppose $\tilde{d}_j \geq \tilde{d}_k$. Then $d^i_j \geq d^i_k$ for all i.

$$\begin{aligned}
\sum_{i=0}^{n-1} \alpha^i z^i_{j,k} &= \sum_{i=0}^{n-1} \alpha^i |d^i_j - d^i_k| \\
&= \sum_{i=0}^{n-1} \alpha^i (d^i_j - d^i_k) \\
&= \tilde{d}_j - \tilde{d}_k = \tilde{z}_{j,k}
\end{aligned}$$

The case when $d_j < d_k$ is analogous. So the LP solution is a convex combination of these $n-1$ cuts. ■

Appendix C: Network inhibition decomposition

Our integer program is the following.

$$(\mathcal{I}) \quad \text{minimize} \quad \sum_{(u,v)\in E} c_{u,v}x_{u,v}$$

$$\text{where} \begin{cases} \sum_{(u,v)\in E} r_{u,v}z_{u,v} \leq B & \\ x_{u,v} + z_{u,v} \geq d_j - d_i & \forall (u,v) \in E \\ x_{u,v} + z_{u,v} \geq d_i - d_j & \forall (u,v) \in E \\ d_s = 0, d_t = 1 & \\ d_v \in \{0,1\} & \forall v \in V \end{cases}$$

Proof of Theorem 8. Consider an arbitrary optimal (fractional) LP solution $\tilde{S} = (\tilde{d}_v, \tilde{x}_{u,v}, \tilde{z}_{u,v})$ for budget B. Sort the d_v values (distance labels of the nodes) so that $\tilde{d}_s \leq \tilde{d}_1 \leq \tilde{d}_2 \leq \cdots \leq \tilde{d}_{n-2} \leq \tilde{d}_t$. Then for each edge (i,j), with $i < j$, we have $\tilde{x}_{i,j} + \tilde{z}_{i,j} = \tilde{d}_j - \tilde{d}_i$. The optimal attack strategy on this fractional cut is the generalization of the integer case: attack the edge with maximum $c_{u,v}/r_{u,v}$ first, pay what it costs to remove it (up to the fraction that the edge is in the cut, namely $\tilde{d}_j - \tilde{d}_i$) and continue until the budget is expended. Thus there will be at most one $(\tilde{x}_{i,j}, \tilde{z}_{i,j})$ pair that does not have one of one of $\tilde{x}_{i,j}$ and $\tilde{z}_{i,j}$ equal to $\tilde{d}_j - \tilde{d}_i$ and the other 0.

Now consider the set of cuts also used in the proof of Theorem 7: s by itself, $\{s, v_1\}$, $\{s, v_1, v_2\}$, etc.). Using the same weights as in that proof (the cut that first includes v_i on the s side has $\alpha_i = \tilde{d}_{i+1} - \tilde{d}_i$), the convex combination gives the proper fractional values for the d_i. Furthermore, edge (i,j) is in cuts $i, i+1, \ldots, j-1$. Thus edge (i,j) is "in the cut" by a fraction $\tilde{d}_j - \tilde{d}_i$ (summing over the contributions of all the integral cuts that include it).

Consider again the fractional solution $\tilde{S}$ with variables $\tilde{x}_{i,j}$ and $\tilde{z}_{i,j}$. These variables give an attack strategy for each of the integral cuts defined above: for cut i, consider all the edges (u,v) in the cut. If $\tilde{z}_{u,v} = |\tilde{d}_v - \tilde{d}_u|$, attack the edge ($z^i_{u,v} = 1$) and if $\tilde{z}_{u,v} = 0$, do not attack it ($z^i_{u,v} = 0$). If (exactly one) edge (u,v) has fractional values for both $\tilde{z}_{u,v}$ and $\tilde{x}_{u,v}$, then partially attack edge (u,v) proportionally. That is $x^i_{u,v} = \tilde{x}_{u,v}/(\tilde{d}_v - \tilde{d}_u)$ and $z^i_{u,v} = \tilde{z}_{u,v}/(\tilde{d}_v - \tilde{d}_u)$.

This setting of the $x^i_{u,v}$ and $z^i_{u,v}$ variables determines a budget B_i for cut i. Because the edges are attacked in order of $c_{u,v}/r_{u,v}$ (cost per unit capacity reduction) within the LP (otherwise $\tilde{S}$ is not optimal), they are also attacked in order for each integral cut, and therefore correspond to optimal attack strategies (for the given budget) for each of the cuts in the decomposition. Using arguments similar to those in the proof of Theorem 7, the convex combination of the d^i_v yields the proper $\tilde{d}_v$ values. Similarly the variables $x_{i,j}$ and $z_{i,j}$ are exactly decomposed. The cost is $\sum \alpha_i B_i$, which is B (exactly corresponding to the attack strategy of the fractional solution) and the residual capacity is $\sum \alpha_i C_i$, where C_i is the residual capacity of cut i when attacked with budget B_i. Intuitively, edge (i,j) is attacked by each cut that includes it by α_i. Each cut expends an α_i fraction of budget B_i on removing the (fractional) edges up to the $B - B_i$ point, and is left with an α_i fraction of its remaining edges to contribute to the residual capacity. ■

Appendix D: s-t path decomposition

Proof of Theorem 10. Relax the integer constraints Q and solve the resulting linear program to get fractional values $\tilde{x}_{u,v}$. Consider the directed graph of edges (u, v) such that $\tilde{x}_{u,v} > 0$ in the fractional solution. We can guarantee that this will have no cycles by using an objective function with positive weights on all edges. (This does not really constrain the objective function: these weights can be chosen small enough to be immaterial.)

Since there are no cycles in this directed graph, do a topological sort and number the vertices according to this sort: s is first, t is last. (Ignore vertices that are not reachable in this graph.) We proceed through the vertices in increasing order of their distance from the root; creating a set of paths for each visited vertex. By the property of topological sort, we would have visited all the predecessors of a vertex before visiting it.

The path set associated with vertex v as a set of s-v paths $p_{v,1}, p_{v,2}, \ldots, p_{v,q(v)}$. Each path $p_{v,i}$ has an associated weight $\alpha_{v,i}$. We maintain the invariant that $\sum_{j=1}^{q(v)} \alpha_{v,j} = \sum_i \tilde{x}_{i,v}$. That is, the sum of the alpha-values associated with the path set at vertex v is equal to the sum of the $\tilde{x}_{i,v}$ values coming in. The path set associated with the source is a single (empty) path $p_{s,1}$ with $\alpha_{s,1} = 1$.

When a vertex is processed, its path set has been computed by its predecessors (as is the case for source s). We then compute its contribution to the path sets of its direct successors. The path set is represented as an arbitrarily-ordered list. Consider the outgoing edges from vertex v. As stated before, we have $\sum_{j=1}^{q(v)} \alpha_{v,j} = \sum_i \tilde{x}_{i,v}$. Place the positive $\tilde{x}_{v,j}$ into an arbitrarily ordered list. Intuitively, think of the $\tilde{x}_{i,v}$ as a stack of boxes with height equal to the capacity at v and think of a similar stack (of equal height) for the $\tilde{x}_{v,j}$. Now slide the two stacks next to each other. This represents a matching of incoming paths to outgoing edges. If there is a boundary between two x boxes then that path is split at that point, part going on the first edge and part on the second. The following pseudocode accomplishes this.

```
$P_0 \leftarrow \{([s], 1)\}$
for $i \leftarrow 0$ to $n - 1$ do
    $P_{i+1} \leftarrow \emptyset$          -- set of paths ending after $v_i$
    $j \leftarrow 0$          -- current box in outgoing stack
    $\beta \leftarrow x_{i,0}$          -- current height remaining in outgoing stack
    forall $(p, \alpha) \in P_i$ do          -- for each box in incoming stack
        if $p$ ends at $v_i$ then
            while $\alpha > \beta$ do
                if $\beta > 0$ then $P_{i+1} \leftarrow P_{i+1} \cup \{(p \cdot v_j, \beta)\}$, $\alpha \leftarrow \alpha - \beta$ fi
                $j \leftarrow j + 1$, $\beta \leftarrow x_{i,j}$
            od
            if $\alpha > 0$ then $P_{i+1} \leftarrow P_{i+1} \cup \{(p \cdot v_j, \alpha)\}, \beta \leftarrow \beta - \alpha$ fi
        else
            $P_{i+1} \leftarrow P_{i+1} \cup \{(p, \alpha)\}$
        fi
    od
od
return $P_n$
```

At the end of this procedure, we have a path set for t. The number of paths in this set is at most $m + 1$, since an incoming path can only be split by the "end" of an outgoing edge in the above procedure.

We begin with a total weight of 1, which is repeatedly divided among the $m + 1$ final paths. When node v is processed, edge (v, j) is given a weight exactly equal to $\tilde{x}_{v,j}$ by construction. So in the decomposition specified by the path set for t, all paths which include edge (u, v) have total weight equal to $\tilde{x}_{u,v}$. Therefore this decomposition is an exact DDA. ■