15-854: Approximations Algorithms **Lecturer:** R.Ravi
Topic: Lagrangean Methods **Date:** Dec. 7 2005
Scribe: Virginia Vassilevska, David Offner

Based on R.Ravi's talk on **Bicriteria Approximations via Lagrangean Relaxations** at the 33rd Annual Conference of the OR Society of Italy.

25.1 Introduction

In this lecture we will consider bicriteria optimization problems - problems in which there are two optimization functions. Many of these problems are NP-hard. We will introduce a notion of approximation for such problems and will use a method called *Lagrangean relaxation* to approximate them.

One of the major goals of computer science has been to tackle computationally hard problems. One clear approach has been to solve these problems exactly, yet it is unclear, especially in the case of NP-hard problems, how to do so in a reasonable amount of time. The OR community has developed interesting techniques but even then some instances can take too long to solve. Another approach has been to try to obtain a feasible solution quickly using some heuristic. When one can prove a performance guarantee for the heuristic we talk of approximation algorithms. The notion of an approximation ratio guarantee has been largely studied when there is a single optimization function. This guarantee is a number $\alpha > 1$ and for maximization problems it states that the ratio of the quality of the obtained solution and the quality of an optimum solution is no more than α. It is similarly defined for minimization problems. When the problem is to find an instance optimizing several functions of the instance, this ratio ceases to make sense.

25.2 Bicriteria Problems

A *bicriteria* problem is a problem formulation P with a set of feasible solutions, $Feasible(P)$ and two functions Obj_1 and Obj_2. P asks for a solution $x \in Feasible(P)$ with optimum (over all $x \in Feasible(P)$) $Obj_1(x)$ and $Obj_2(x)$. P specifies what *optimum* means - it could be max or min, and it could be different for Obj_1 and Obj_2. Often the problems are phrased as given bounds B_1 and B_2 for Obj_1 and Obj_2 to find x satisfying the bounds. The Obj_2 constraint is called the 'complicating' constraint. WLOG P is defined as follows:

Find x s.t.

$$\begin{aligned} Obj_1(x) &\le B_1 \\ Obj_2(x) &\le B_2 \\ x &\in Feasible(P) \end{aligned}$$

An (α_1, α_2)-*bicriteria approximation algorithm* outputs a solution x' to a bicriteria problem P s.t.

$$
\begin{aligned}
Obj_1(x') &\leq \alpha_1 B_1 \\
Obj_2(x') &\leq \alpha_2 B_2 \\
x' &\in Feasible(P)
\end{aligned}
$$

Notice that the above definitions clearly extend the notion of an approximation algorithm, and can be extend to a definition of an approximation algorithm for an n-criteria problem with $n > 2$.

We will investigate three examples: Two Cost Minimum Spanning Tree, Minimum-k-Cut and Degree Bounded Minimum Spanning Tree.

25.3 Two Cost Min Spanning Tree

Two Cost Minimum Spanning Tree (2MST): Given a graph $G = (V, E)$, nonnegative functions $c : E \to \mathbb{Z}^+$, $\ell : E \to \mathbb{Z}^+$ and budgets $C \geq 0$ and $L \geq 0$, find a spanning tree T with $c(T) \leq C$ and $\ell(T) \leq L$.

25.3.1 NP-hardness of 2MST

If C and L are given in unary, 2MST can be easily solved in Ptime using dynamic programming. If however C and L are given in binary, the problem is NP-hard even on on series-parallel graphs. (Such problems are called *weakly NP-hard.*)

To show NP-hardness we will give a reduction from the Partition problem.
Partition: Given numbers $a_1, \ldots, a_n \in \mathbb{Z}$, does there exist $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = \sum_{j \notin I} a_j$?

Consider the following figure:

There are n vertices u_i for $i = 1, \ldots, n$, n vertices d_i for $i = 1, \ldots, n$, $n+1$ vertices c_i for $i = 1, \ldots, n+1$. Each edge (c_i, u_i) has $c(c_i, u_i) = a_i$ and $\ell(c_i, u_i) = 0$. Picking this edge in the spanning tree corresponds to putting i into I for Partition. Each edge (c_i, d_i) has $c(c_i, u_i) = 0$

and $\ell(c_i, u_i) = a_i$. Picking this edge in the spanning tree corresponds to **not** putting i into I for Partition.

Each edge (u_i, c_{i+1}) has $c(u_i, c_{i+1}) = 0$ and $\ell(u_i, c_{i+1}) = 0$. Each edge (d_i, c_{i+1}) has $c(d_i, c_{i+1}) = 0$ and $\ell(u_i, d_{i+1}) = 0$. Either of these two edges can be added to complete a subgraph to a tree without increasing c or ℓ.

Set $C = L = \frac{\sum_i a_i}{2}$.

Suppose Partition has a solution I. Then build a spanning tree T as follows: for every $i \in I$ put (c_i, u_i) and (u_i, c_{i+1}) in T; for every $i \notin I$, put (c_i, d_i) and (d_i, c_{i+1}) in T. Clearly, $c(T) = \sum_{i \in I} a_i = C$ and $\ell(T) = \sum_{i \notin I} a_i = L$.

Now suppose there is a spanning tree T with $c(T) \leq C$ and $\ell(T) \leq L$. Since T is a spanning tree, for every i, either (c_i, u_i) or (c_i, d_i) is in T, or both. If it contains both, then either (u_i, c_{i+1}) or (d_i, c_{i+1}) is in T, but not the other. If (u_i, c_{i+1}) (or (d_i, c_{i+1})) is not in T then we can remove (c_i, u_i) (or (c_i, d_i)) from T to obtain an even better solution to 2MST.

If both (u_i, c_{i+1}) and (c_i, u_i) are in T (hence $(c_i, d_i) \notin T$), then put i into I. Otherwise, both (d_i, c_{i+1}) and (c_i, d_i) are in T (hence $(c_i, u_i) \notin T$), and we don't include i in I.

Consider

$$\sum_{i \in I} a_i = \sum_{i:(c_i, u_i) \in T} a_i = c(T) \leq C = \sum_i a_i/2.$$

Also

$$\sum_{i \notin I} a_i = \sum_{i:(c_i, d_i) \in T} a_i = \ell(T) \leq L = \sum_i a_i/2.$$

Since $\sum_i a_i = \sum_{i \in I} a_i + \sum_{i \notin I} a_i \leq \sum_{i \in I} a_i + \sum_i a_i/2$, $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$. And I is a valid solution to Partition.

25.4 Min k-Cut

Min k-Cut: Given a graph $G = (V, E)$, function $s : E \to \mathbb{Z}^+$, budget $S \geq 0$, integer $k > 1$, $Feasible(\text{Min } k\text{-Cut}) = \{A | A \subseteq E\}$. Find $A \in Feasible(\text{Min } k\text{-Cut})$ such that the number of connected components of $G \setminus A$ is at least k.

The two constraints for this problem are the number of connected components and the cost of the edges removed.

25.4.1 NP-hardness for Min k-Cut

It seems that the reduction given in class doesn't quite work. So we will present a proof given by Downey *et.al* in [8].

The reduction is from k-Clique. Given $G = (V, E)$, an instance of k-clique, create an instance of Min K-cut as follows. Let $K = k + 1$ and $S = k(n^2 + n) + (k - 1)\binom{k}{2}$. Create $n + 2$ cliques of size n^4 named C_{top}, C_{bottom} and C_v for each $v \in V$. These are all vertices of the graph $G' = (V', E')$ we are creating. There are $n^4(n + 2)$ vertices in V'. The edges are as follows. There is exactly one edge between every C_u and C_v for $(u, v) \in E$ (it doesn't matter between which vertices of C_u and C_v). For C_{top} and any C_v, there is a matching from some $\binom{k}{2}$ vertices of C_v to some $\binom{k}{2}$ vertices of C_{top}. For C_{bottom} and any C_v, there is a matching from some $n^2 + n - deg(v)$ vertices of C_v to some $n^2 + n - deg(v)$ vertices of C_{top}. For any $e \in E'$, $s(e) = 1$.

Now suppose G has a k-clique C. To cut $\{C_u | u \in C\}$ from C_{bottom} one needs to remove $k(n^2 + n) - \sum_{u \in C} deg(u)$ edges. To cut $\{C_u | u \in C\}$ from C_{top} one needs to remove $k\binom{k}{2}$ edges. To cut the C_u for $u \in C$ from the rest of the C_v and from each other, one needs to remove

$$\sum_{u \in C} deg(u) - \text{ number of edges in } C = \sum_{u \in C} deg(u) - \binom{k}{2}.$$

Altogether the size of this $k + 1$ cut is

$$(k(n^2 + n) - \sum_{u \in C} deg(u)) + k\binom{k}{2} + (\sum_{u \in C} deg(u) - \binom{k}{2}) = k(n^2 + n) + (k - 1)\binom{k}{2} = S.$$

Now suppose there is a K-cut A with at most S edges in G'. For large n, $S < n^4 - 1$ and hence A must keep each large clique in the same connected component.

Furthermore, because G' can be split into K components using only S edges, it must be possible to accomplish this by creating $K - 1 = k$ connected components in the part of G' which is a copy of G (but with cliques instead of vertices) and letting the rest be the last connected component.

Now consider the k connected components that are split from C_{top} and C_{bottom}. Suppose one of them C' contains $v_1, \ldots, v_r$ for $r > 1$. Then the number of edges that need to have been removed is

$$r\binom{k}{2} + r(n^2 + n) - \sum_{v_i \in C'} deg(v_i) + |\{(v_i, u) \in E | v_i \in C', u \notin C'\}|.$$

Now suppose we move v_1 to the component containing C_{top} and C_{bottom}. The change in the number of edges removed is

$$-\binom{k}{2} - (n^2 + n) + deg(v_1) - |\{(v_1, u) \in E | u \notin C'\}| < 0.$$

Hence in a min K-cut all components except for the ones containing C_{top} and C_{bottom} contain exactly one C_v. Let C consist of k components not containing C_{top} or C_{bottom} The number of edges used by C is

$$(k(n^2 + n) - \sum_{u \in C} deg(u)) + k\binom{k}{2} + \sum_{u \in C} deg(u) - \text{ number of edges in } C \leq k(n^2 + n) + (k - 1)\binom{k}{2}.$$

Hence number of edges in $C \geq \binom{k}{2}$ and C must be a k-clique.

Thus Min k-cut is NP-hard. This also shows it is W[1] hard. Nothing is known about the approximation hardness of the problem.

25.5 Degree Bounded MST (DBMST)

Degree Bounded MST (DBMST): Given $G = (V, E)$, $Feasible$(DBMST) consists of all spanning trees of G. One is also given a degree bound $B > 0$, a cost function $c : E \to \mathbb{R}^+$ and a cost bound $C > 0$. One seeks a best spanning tree T such that $deg_T(v) \leq B \quad \forall v \in V$ and $\sum_{e \in T} c(e) \leq C$.

25.5.1 NP-hardness for DBMST

The problem is NP-hard even for $B = 2$. The following is a reduction from TSP. Start with an instance of TSP, $(G = (V, E), c, C)$. For every edge $e = (u, v)$, add two new vertices w_u and w_v, replace e with edges (w_u, u) and (v, w_v) with $c(w_u, u) = c(e)/2 = c(v, w_v)$ and do DBMST for this new graph with $B = 2$ and C. Take the best cost DBMST over all edges and convert it to a tour by replacing the two edges incident to w_u and w_v by (u, v).

25.6 Approximation

To obtain approximation algorithms for multicriteria optimization problems, researchers have tried several approaches. One approach is the so called goal programming. It ranks the objective functions in some way and then attempts to optimize sequentially, *i.e.* over all feasible solutions which are best for the objectives up to objective i, find all solutions that are best for objective $i + 1$ etc. Another approach uses the notion of Pareto optimality. It finds a solution such that one cannot improve any one objective function without decreasing the quality with respect to another objective. An optimum solution is clearly Pareto optimal. Yet for many problems the set of Pareto optimal solutions can be large and their quality can differ a lot.

We have extended the notion of approximation ratio to Bicriteria budget problems. An (α_1, α_2)-approximation algorithm for P

$$\begin{aligned} Obj_i(x) \leq B_i,\ i = 1, 2 \\ x \in feasible(P) \end{aligned}$$

gives a solution x' with

$$\begin{aligned} Obj_i(x') \leq \alpha_i B_i,\ i = 1, 2 \\ x' \in feasible(P). \end{aligned}$$

25.7 Lagrangean relaxation

The idea of a Lagrangean relaxation is to convert a bicriteria problem into a single criterion one by varying a penalty parameter λ. More formally, given a problem P

$\min Obj_1(x)$ s.t.

$$Obj_2(x) \leq B$$
$$x \in feasible(P)$$

one converts it into a Lagrangean relaxation $L(\lambda)$:

$\min Obj_1(x) + \lambda(Obj_2(x) - B)$ s.t.

$$x \in feasible(P)$$

The parameter λ is a penalty in the sense that if the Obj_2 constraint in the original problem is not satisfied then the new objective is larger than it could be - think of λ being huge.

We can easily obtain the following.

Lemma 25.7.1 *Opt(P) is at least* $\max_{\lambda > 0} L(\lambda) = LR^*$.

The proof is trivial - an optimum feasible solution x^* of P will have value $Obj_1(x^*) \geq Obj_1(x^*) + \lambda(Obj_2(x^*) - B) \geq L(\lambda)$ for all $\lambda > 0$.

Lagrangean relaxation and λ values often provide good lower bounds and suggest approximation algorithm strategies. Often, as is the case with 2MST, the relaxation itself is a good approximation algorithm. In the k-cut application, the Lagrangean formulation suggests a good approximation algorithm. As in the DBMST application, optimal Lagrange multipliers, under a dual interpretation, can aid in the design and analysis of bicriteria approximations.

25.8 Lagrangean Relaxation for 2MST

2MST was the problem that seeks to minimize $\sum_e c(e)x_e$ s.t. $\sum_e \ell(e)x_e \leq L$, $x_e \in \{0,1\}$, and the edges with $x_e = 1$ define a spanning tree of G. A Lagrangean formulation $L(z)$ for 2MST is as follows. Here we use z instead of λ.

$\min \sum_e c(e)x_e + z(\sum_e \ell(e)x_e - L)$ s.t.

$$x_e \text{ forms a spanning tree of } G$$

The main observation is that $L(z)$ is a minimum spanning tree problem with weights $w(e) = c(e) + z\ell(e)$. Hence one can solve any Lagrangean subproblem $L(z)$ in ptime. We now characterize the solution space as function of z.

Let for a fixed T, $L_z(T) = c(T) + z(\ell(T) - L)$. $L_z(T)$ is a linear function of z with slope $(\ell(T) - L)$ and intercept $c(T)$. The slope is nonpositive for feasible solutions of the 2MST instance. To visualize the Lagrangean relaxation values $L(z)$, plot the L_z values of all trees vs z and take the minimum over all trees for each z. The Lagrangean values form the lower envelope. It is concave and piecewise linear. The motivation comes from the lemma: if T is optimum for some z^* Lagrangean relaxation and the T line going through z^* has 0 slope then T is optimal for 2MST.

Fact 25.8.1 *Fix $w_{z^*} = c + z^*\ell$. The set of MSTs according to w_{z^*} can be ordered according to slope $T_0, T_1, \ldots$ s.t. every consecutive pair differs by single edge swap.*

Since $\ell(T_0) \geq L$ and $\ell(T_k) \leq L$ there is i such that $\ell(T_i) \geq L$ and $\ell(T_{i+1}) \leq L$. Moreover, since these two trees differ by one edge swap, $\ell(T_i) \leq L + \ell_{max}$, where $\ell_{max} = \max_e \ell(e)$.

To obtain a $(1,2)$-approximation, delete all edges of legth $> L$ so that the maximum edge legth remaining is $\ell_{max} \leq L$. Find z^* and a pair of trees T_i and T_{i+1} both optimal at z^* with $\ell(T_{i+1}) \leq L$ and $L \leq \ell(T_i) \leq L + \ell_{max} \leq 2L$.

Claim 25.8.2 *Let C^* and LR^* be the values of the optimum 2MST solution and the optimum Lagrangean relaxation respectively. Then $c(T_i) \leq LR^* \leq C^*$.*

Proof: Since $\ell(T_i) \leq L$ and $C^* \geq LR^*$,

$$c(T_i) = c(T_i) + z^*(\ell(T_i) - L) - z^*(\ell(T_i) - L) = L(z^*) - z^*(\ell(T_i) - L) \leq LR^* \leq C^*.$$

■

In summary, $c(T_i) \leq C^*$ and $\ell(T_i) \leq 2L$ and we obtain a $(1,2)$ approximation.

25.8.1 A PTAS for 2MST

Ravi and Goemans [9] give a PTAS for 2MST. Given ϵ, the algorithm proceeds as follows:

- Delete all edges of length $> \epsilon L$.
- Exhaustively consider all sets of up to n deleted edges.
- For each combination, complete the tree using the (1,2)-approximation.

The algorithm runs in poly-time (in $n, 1/\epsilon$) since the second step requires $O(n^{O(\frac{1}{\epsilon})})$ iterations. Note that the output tree still has cost $\leq C^*$, and length $\leq (1+\epsilon)L$, since $l_{max} \leq \epsilon L$ in the modified graph.

25.8.2 Running Time Details

In the (1,2)-approximation algorithm, the number of breakpoints in L_z is trivially bounded by the number of possible edge swaps $= \binom{m}{2}$, by Fact 25.8.1. Better bounds are known. Megiddo and Gusfield give an algorithm to compute the next breakpoint from the previous one in time of the same order as computing the Lagrangean subproblem–an MST computation. Running time improvements are possible using ideas from Megiddo and a parallel MST algorithm of Coleman.

25.9 A (1,2)-approximation for k-Cut

Let $\kappa(A)$= the number of connected components in $G' = (V, E \setminus A)$. Then we can formulate k-Cut as

$$\min \sum_{e \in A} s(e)$$
$$s.t. \quad \kappa(A) \geq k$$

The Lagrangean Relaxation is

$$\min\ s(A) + b(k - \kappa(A)) \equiv \min\ s(A) - b(\kappa(A) - 1) + b(k-1)$$

For fixed b,

$$g(b) = \min_{A \subset E} \quad s(A) - b(\kappa(A) - 1)$$

is called the attack problem and Cunningham [1] proved it is solvable in poly-time. In his paper he proves:

Lemma 25.9.1 *$\kappa(b)$ for the optimal subset of edges for b is non-decreasing in b.*

In addition, since for any fixed A, the slope of $g(b)$ is $-b(\kappa(A) - 1)$ (where $\kappa(A) \in [n]$), we have

Lemma 25.9.2 *$g(b)$ is continuous, non-increasing, concave, piecewise linear, and has no more than $n - 1$ breakpoints.*

Now note that if for some b, $g(b)$ is minimized at a cut A with $\kappa(A) = k$, then A is an optimal k-cut. Otherwise we seek a breakpoint in the $g(b)$ curve where the number of connected components goes from $< k$ to $> k$. We will combine these two solutions to get a k-cut with cost at most twice the optimal.

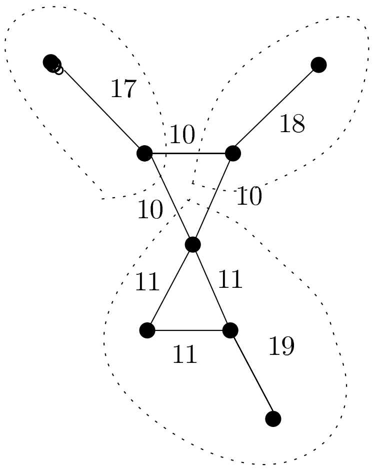
The following theorem gives us the last piece of structure of the $g(b)$ curve we will need:

Theorem 25.9.3 *If b_0 is a breakpoint of $g(b)$ induced by edge sets A and B where $\kappa(A) > \kappa(B)$, then $B \subset A$.*

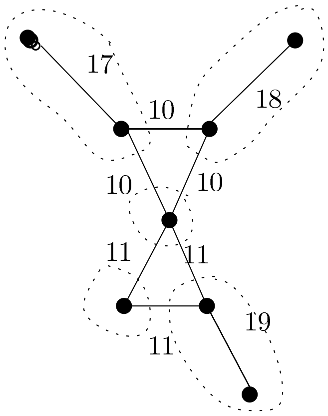
Corollary 25.9.4 *If b_0 is a breakpoint induced by edge sets A and B where $\kappa(A) > \kappa(B)$, then $A \setminus B$ is contained in some connected component of $G' = (V, E \setminus B)$.*

Ravi and Sinha [10] gave the following (1,2)-approximation algorithm for k-cut:

- Construct $g(b)$ fully. Let the breakpoints be $b_0, b_1, \ldots, b_l$, induced by edge sets $\emptyset = A_0, A_1, \ldots, A_l$.
- If there exists i such that $\kappa(A_i) = k$, then A_i is the optimal solution.
- Otherwise, identify i such that $\kappa(A_i) < k < \kappa(A_{i+1})$.

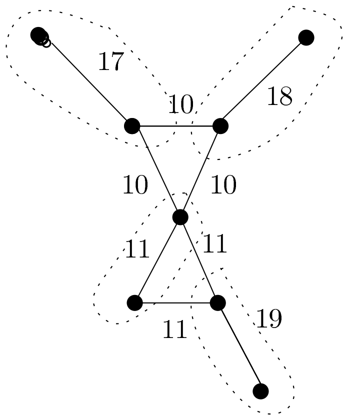


Optimal Strength
=30/(3-1) = 15



Next Optimal Strength
= 33/(3-1) = 16.5

5-cut–too much!



Pick 1 "cheapest shore"
Total Cost = 52

- Pick A_i. Pick the $k - \kappa(A_i)$ "cheapest shore" components from A_{i+1}, and call this set of edges B.
- Return $A_i \cup B$

E.g. In the figure, we seek a 4-cut. We go from a 3-cut to a 5-cut at the breakpoint. So the algorithm chooses the 3-cut with one additional shore from the 5-cut.

To analyze the performance of the algorithm, recall that if we do not find an optimal k-cut, the i chosen is such that $\kappa(A_i) < k < \kappa(A_{i+1})$. Since an optimal k-cut A is sub-optimal for attack than A_i and A_{i+1}, we derive

$$s(A) > \left(\frac{\kappa(A_{i+1}) - k}{\kappa(A_{i+1}) - \kappa(A_i)}\right) s(A_i) + \left(\frac{k - \kappa(A_i)}{\kappa(A_{i+1}) - \kappa(A_i)}\right) s(A_{i+1})$$

In our solution, we pay $s(A_i)$ for edges in A_i, and the cheapest $k - \kappa(A_i)$ shores from $A_{i+1} \setminus A_i$. The total cost of all shores in $A_{i+1} \setminus A_i$ is exactly $2(s(A_{i+1}) - s(A_i))$. Hence our cost cost is

$$\left(\frac{2(\kappa(A_{i+1}) - k)}{\kappa(A_{i+1}) - \kappa(A_i)} - 1\right) s(A_i) + 2\left(\frac{k - \kappa(A_i)}{\kappa(A_{i+1}) - \kappa(A_i)}\right) s(A_{i+1}) < 2s(A)$$

yielding a performance ratio of 2.

25.10 A Lagrangean Approximation Algorithm for DBMST

Formulate DBMST as

$$\begin{aligned} \min \quad & c(T) \\ s.t. \quad & T \text{ is a spanning tree} \\ & \delta_T(v) \leq B \quad \forall v \in V \end{aligned}$$

Write the Lagrangean relaxation as

$$\begin{aligned} \min \quad & c(T) + \sum_{v \in V} z_v(\delta_T(v) - B) \\ s.t. \quad & T \text{ is a spanning tree} \end{aligned}$$

The main observation is that the Lagrangean subproblem is MST under weights $c^z_{uv} = c_{uv} + z_u + z_v$.

Based on a local search algorithm, Furer and Raghavachari [2] and Fischer [3] gave an approximation algorithm which outputs an MST with degree at most $r \cdot \Delta^* + \lceil \log_r n \rceil$ where Δ^* is the minimum max-degree of any MST for any $r > 1$. We can apply this algorithm to the edge-weights obtained from the Lagrange multipliers c^z_{uv}. Since the performance ratio of the local improvement algorithm is proved via a lower bound that can be interpreted as a LP dual to the minimum max-degree MST problem, we hope to show that the resulting tree has a low max-degree.

Könemann and Ravi [11] showed that this idea works, with the following modification: Solve for the optimal Lagrange multipliers for slightly weakened degree bounds. In particular:

- Find optimal Lagrange multipliers z^*_v's for $deg_T(v) \leq (1+\epsilon)B \quad \forall v \in V$.
- Output MST under costs $c^z_{uv} = c_{uv} + z^*_u + z^*_v$ of approximately minimum max-degree.

We can prove bounds on max-degree and cost using the optimality of Lagrangean multipliers used by the algorithm:

The output tree has degree at most $r(1+\epsilon)B + \lceil \log_r n \rceil$, by optimality of the Lagrangean multipliers used in the modified edge-weights.

The output tree has cost at most $(1 + 1/\epsilon)$ times minimum by bounding away the contribution of the Lagrangean multiplier terms using a scaling argument based on extra slack (ϵB) in the degree constraints.

25.11 Other Applications and Open Problems

Lagrangean Relaxations have also been useful in adapting spproximation algorithms for an original problem to a closely related budget problem. In particular:

- Application of approximation algorithms for Uncapacitated Facility Location to design one for the k-median problem, both under metric costs (Jain and Vazirani '98 [4])
- Application of approximation algorithm for prize-collecting Steiner trees to design one for minimum-cost k trees (Blum, Ravi, and Vempala '95 [5], Garg '96 [6], Chudak, Roughgarden and Williamson '98 [7])

Open Problems:

- Are there natural applications of Lagrangean relaxation to new problems? CAn any existing results be recast in this framework?
- How can one tackle situations where the subproblem is NP-hard? E.g. degree-bounded minimum-cost Steiner trees. A direct primal-dual approach that updates duals and Lagrange multipliers while making progress on both objectives is promising.

References

[1] Cunningham, W.H. 1985. Optimal attack and reinforcement of a network. *Journal of the ACM* 32(3):549-561.

[2] Martin Fürer and Balaji Raghavachari. Approximating the minimum-degree Steiner tree to within one of optimal. *Journal of Algorithms*, 17(3):409-423, November 1994.

[3] T. Fischer. Optimizing the degree of minimum weight spanning trees. Technical Report TR 93-1338, Dept. of Computer Science, Cornell University, Ithaca, NY 14853, 1993.

[4] Kamal Jain and Vijay V. Vazirani. Spproximation Algorithms for Metric Facility Location and k-Median Problems Using the Primal-Dual Schema and Lagrangean Relaxation. *Journal of the ACM* 48(2):274-296, March 2001.

[5] Avrim Blum, R.Ravi, and Santosh Vempala. A Constant-Factor Approximation Algorithm for the k-MST Problem. *J. Comput. Syst. Sci.*, 58(1):101-108, 1999.

[6] Naveen Garg. A 3-Approximation for the Minimum Tree Spanning k Vertices. FOCS, 1996, 302-309.

[7] Fabian A. Chudak, Tim Roughgarden, David P. Williamson. Approximate k-MSTs and k-Steiner Trees via the primal-dual method and Lagrangean relaxation. *Mathematical Programming*, Volume 100, issue 2, Jun 2004, 411-421.

[8] Rodney G. Downey, Vladimir Estivill-Castro, Michael R. Fellows, Elena Prieto, Frances A. Rosamond: Cutting Up is Hard to Do: the Parameterized Complexity of k-Cut and Related Problems. Electr. Notes Theor. Comput. Sci. 78: (2003)

[9] Ravi, R. and Goemans, M. X. 1996. The Constrained Minimum Spanning Tree Problem (Extended Abstract). In Proceedings of the 5th Scandinavian Workshop on Algorithm theory (July 03 - 05, 1996). R. G. Karlsson and A. Lingas, Eds. Lecture Notes In Computer Science, vol. 1097. Springer-Verlag, London, 66-75.

[10] R. Ravi, Amitabh Sinha II: Approximating k-cuts via network strength. SODA 2002: 621-622

[11] Jochen Könemann, R. Ravi: A matter of degree: improved approximation algorithms for degree-bounded minimum spanning trees. STOC 2000: 537-546