CMSC 858K Advanced Algorithms	Lecture 25
Lecturer: Samir Khuller	May 6, 1997

Notes by Josh Burdick.

14 Using duality to analyze (and design) approximation algorithms

This material is covered in Chapter 4 (which was written by Goemans and Williamson) of the *Approximation Algorithms* book edited by Hochbaum. (The original paper appeared in SICOMP 1995.)

14.1 Using duality to bound OPT-IP

Duality was first studied as a way of proving optimality of solutions to an LP instance. It also played a role in the minimum-weight bipartite perfect matching algorithm, where the labeling function is essentially the dual problem, although the concept of a "dual problem" was not formulated at that time.

Assume there is some integer program (IP) that you wish to solve; assume without loss of generality that you wish to minimize the value of the objective function. Let OPT-IP denote the value of the objective function at the optimal solution.

Note that the value of the LP relaxation of this problem can be no more than OPT-IP, since the optimal IP solution is also a feasible LP solution – in general, OPT-LP is less than OPT-IP. By the Weak Duality theorem of linear programming, any feasible dual solution is a lower bound on OPT-LP (and thus OPT-IP).

To summarize, we have that

$$\text{any solution to the } \textit{dual} \text{ LP} \quad \leq \quad \text{OPT-DLP} = \text{OPT-LP} \quad \leq \quad \text{OPT-IP}$$

(This is assuming that the primal IP is a minimization problem; if it were a maximization problem, all the inequalities would be reversed.)

Using this to bound approximation factors Our main goal will be to produce a feasible solution whose cost is at most α times the value of the dual feasible solution that we can find, which is also a lower bound on the cost of the optimum solution.

The Steiner tree problem The Steiner tree problem is: Given an undirected graph with weights on the edges, and a subset T of the vertices, find a minimum-weight tree that connects all the vertices in T. (The tree needn't connect all the vertices in V, but it optionally can use vertices from $V - T$, if that will help to connect the vertices in T.)

The problem is NP-complete. There was a sequence of results, with improving approximation ratios:

- Kou, Markowsky and Berman discovered a 2-approximation (this was homework 3, problem 3)
- Takahashi and Matsuyama proved that a simple greedy algorithm also gives a 2 approximation.
- Zelikovsky obtained an $\frac{11}{6} \approx 1.833$ approximation ratio by considering three vertices at a time, and determining whether or not Steiner nodes would help to connect just those three. (Steiner nodes are vertices in $V - T$.)
- Berman and Ramaiyer improved this bound to $\frac{16}{9} \approx 1.777$ with a more complicated algorithm which considers more vertices at a time.
- Karpiniski and Zelikovsky later found a bound of 1.644.
- Agrawal, Klein, and Ravi studied a generalization of the Steiner tree problem, the pairwise connection problem. Here, given some sets of vertices $\{s_1, s_2, ...s_n\}$ and $\{t_1, t_2, ...t_n\}$, the goal is to pick edges so that s_i is connected to t_i for all i such that $1 \leq i \leq n$.

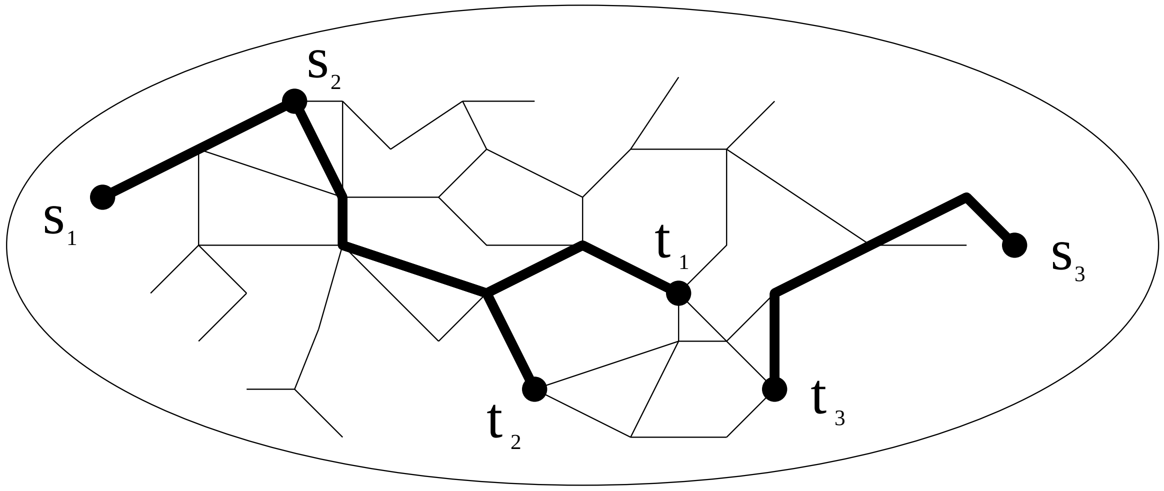

Figure 3: The pairwise connection problem.

Goemans and Williamson found that the generalized Steiner tree problem, and a variety of other problems (mostly network-design problems), could all be stated and approximated in a unified framework.

14.2 The "proper function" framework

Now we return to Goemans and Williamson's general approximation framework. We are given a graph $G = (V, E)$, with weights on the edges. We will have indicator variable x_e for each edge, in the usual way. Also, for any subset of vertices S, let $\delta(S)$ be the set of all edges from S to $V - S$. Let f be a function from subsets (also, in this context, called "cuts") of the vertices V to $\{0, 1\}$.

A problem is expressible in this framework if you can write it as an integer programming problem:

$$\text{minimize} \sum w_e \cdot x_e$$

subject to the constraints

$$\sum_{e \in \delta(S)} x_e \geq f(S)$$

This looks a lot like the usual integer programming formulation. The main difference is the use of the function f. Intuitively, you want to pick f to be 1 whenever you want to ensure that there's at least one edge crossing some cut S.

In the dual problem, we have a variable for every cut S of V. The dual problem is

$$\text{maximize} \sum y_S \cdot f(S)$$

subject to the constraints

$$\sum_{S: e \in \delta(S)} y_S \leq w_e$$

There are $2^{|V|}$ different cuts, and therefore $2^{|V|}$ different dual variables y_S. Fortunately, most of them are zero, and we need only keep track of the nonzero dual variables.

The function f is called a "$\{0, 1\}$ proper function" (to be defined formally later).

14.3 Several problems, phrased in terms of proper functions

Many problems related to network design can be stated in these terms.

Generalized Steiner tree We are given an undirected, weighted graph, and two sets of vertices, $\{s_1, s_2, ...s_n\}$ and $\{t_1, t_2, ...t_n\}$. The goal is to find a minimum-weight subset of edges such that s_i is connected to t_i, for $1 \leq i \leq n$.

To solve this, let $f(S) = 1$ if for some i, $s_i \in S$ and $t_i \notin S$ (or vice versa); otherwise let $f(S) = 0$. This forces s_i to be connected to t_i.

Steiner tree Let T be the set of terminal nodes (that is, nodes which are required to be connected.) Let S be a cut of V. Define

$$\begin{aligned} f(S) &= 1 \text{ if } S \neq \emptyset,\ S \cap T \neq T, \text{and } S \subset V \\ &= 0 \text{ otherwise} \end{aligned}$$

If a cut S contains at least one of the terminal nodes, but not all of them, then there should be at least one edge across the cut, to connect with the terminal nodes not in S.

Minimum-weight perfect matching Let S be a cut of V. Let $f(S) = 1$ iff $|S|$ is odd; otherwise let $f(S) = 0$. This forces each tree in the forest to have even size. Once we have these even components, we can convert it into a matching of the same size.

14.4 The greedy algorithm

Here is Goemans and Williamson's greedy algorithm to 2-approximate problems stated in the proper function framework:

1. Initialize the set of connected components $C = V$.
2. Increase all of the y_Ss that haven't hit a constraint, all at once. If a constraint hits equality, pick the edge corresponding to that constraint. A new connected component S' has been created, so add S' to C, and create a new variable $y_{S'}$ for it. $y_{S'}$ starts at 0.
3. Repeat step 2 until enough edges are picked.
4. Afterwards, throw away any unneeded edges. To do this, let A and B be the connected components at each end of some edge. If $f(A) = 0$ and $f(B) = 0$, you can discard that edge.

At the outset, there are $|V|$ seperate connected components, each consisting of one vertex. Each connected component has a variable y_S associated with it, which grows as the algorithm progresses. If a constraint is hit, that y_S stops growing. Later, of course, the vertices in y_S may be added to some new connected component S'.

The cost of the solution found is no more than two times the cost of some dual solution; by the arguments at the start of the lecture, that means that the cost of the solution found is within a factor of two of optimal. (The proof will be given next lecture.)

An example run of the algorithm Here is a small example of how this algorithm runs. The problem being solved is minimum perfect matching. The vertices are on a plane, and the weights between them are Euclidean distances.

The primal problem is to meet the constraints, and find a minimum perfect matching, using a minimum-weight set of edges. The dual variables have been described by Jünger and Pulleyblank as "moats", or regions surrounding the connected components. They note that in the case of the minimum-weight perfect matching problem in the Euclidean plane, the dual problem

> "...attempts to find a packing of non-overlapping moats that maximizes the sum of the width of the moats around odd-sized sets of vertices. The algorithm... can thus be interpreted as growing odd moats at the same speed until two moats collide, therefore adding the corresponding edge, and repeating the process until all components have even size." (Hochbaum, p. 173.)

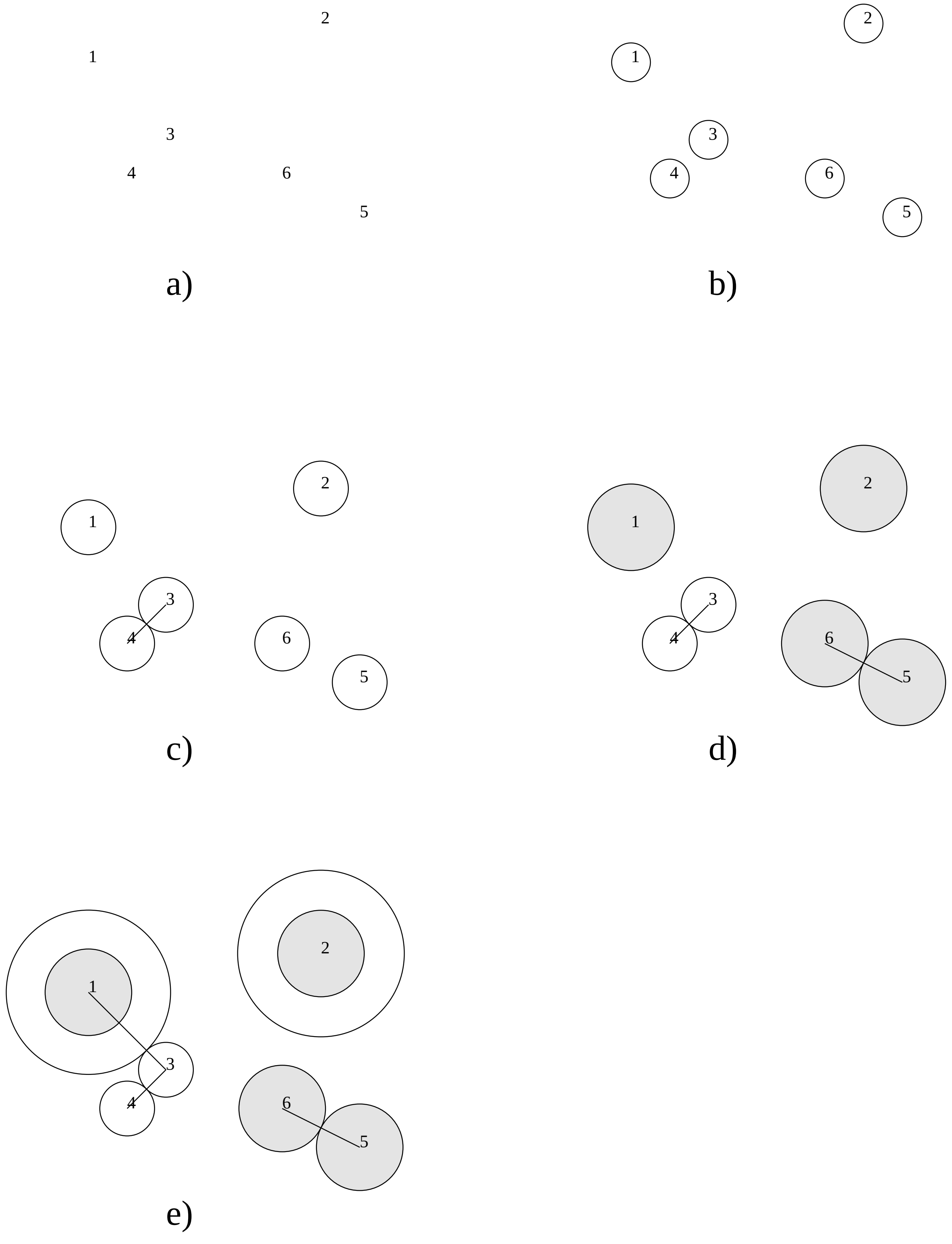

Figure 4: The greedy algorithm for minimum-weight perfect matching in the Euclidean plane (example from class.)

The diagram on p. 174 of Hochbaum illustrates this; the example from class was similar.

Fig.$\tilde{3}$ a) shows six points in the Euclidean plane. The values of the dual variables $y_{\{1\}}$, $y_{\{2\}}$, $y_{\{3\}}$, $y_{\{4\}}$, $y_{\{5\}}$, and $y_{\{6\}}$ are all initialized at zero.

In b), the dual variables have increased some, but no constraint has yet been hit.

In c), dual variables $y_{\{3\}}$ and $y_{\{4\}}$ have hit a constraint. So, they are connected with an edge. A new connected component has been formed. We needn't create a new dual variable, $y_{\{3,4\}}$, because that variable will always be zero.

In d), the dual variables $y_{\{1\}}$, $y_{\{2\}}$, $y_{\{5\}}$, and $y_{\{6\}}$ have all increased, and an edge is added from 5 to 6. Vertices 5 and 6 are now "inactive." At this point, only vertices 1 and 2 are growing.

In e), vertex 1 gets connected to vertices 3 and 4. The dual variable $y_{\{1,3,4\}}$ is formed, and will keep increasing.

When no dual variables are active anymore (that is, when every connected component has an even number of vertices), this phase will stop. The algorithm will now delete as many edges as it can.