CMSC 858K Advanced Algorithms	Lecture 26
Lecturer: Samir Khuller	May 13, 1997

Notes by Thomas Vossen.

15 A General Approximation Technique for Constrained Forest Problems

Today's lecture notes continue with the general framework for approximation algorithms that was started last time. We discuss the algorithm of Goemans and Williamson in more detail, and proof its correctness and approximation bound. The material covered here can also be found in Goemans and Williamson's article (from handout).

15.1 Constrained Forest Problems

Following the terminology of Goemans and Williamson, we call the problems to which the framework can be applied *constraint forest problems.* Given a graph $G = (V, E)$, a function $f : 2^V \to \{0,1\}$ and a non-negative cost function $c : E \to \mathcal{Q}_+$, the associated constrained forest problem is then defined by the following integer program (IP):

$$\begin{array}{lll} \text{Min} & \sum_{e \in E} c_e x_e & \\ \text{subject to:} & & \\ & \sum_{e \in \delta(S)} x_e \geq f(S) & S \subset V, S \neq \emptyset \\ & x_e \in \{0,1\} & e \in E \end{array}$$

where $\delta(S)$ denotes the set of edges having exactly one endpoint in S. Any solution to integer program corresponds to a set of edges $\{e \in E \mid x_e = 1\}$. Observe that this set of edges can always be turned into a forest by arbitrarily deleting an edge from a cycle (By definition of $\delta(S)$, this does not affect feasibility).

We only consider *proper* functions f here, that is, functions $f : 2^V \to \{0,1\}$ for which the following properties hold:

1. **Symmetry** $f(S) = f(V - S)$ for all $S \subseteq V$;
2. **Disjointness** If A and B are disjoint, then $f(A) = f(B) = 0$ implies $f(A \cup B) = 0$.

Furthermore, it is assumed that $f(V) = 0$. We have already seen several examples of graph problems that can be modeled as proper constraint forest problems. Some additional examples are the following :

- **Minimum Spanning Tree problem** Define $f(S) = 1$ for all subsets S.
- **Shortest $s - t$ Path problem** Define $f(S) = 1$ *iff* $|S \cap (s,t)| = 1$.

The greedy algorithm of Goemans and Williamson can be seen as a primal-dual algorithm, in which an approximate solution to the primal and to the LP-relaxation of the dual are constructed simultaneously, such that the primal solution value is at most a constant times the dual solution value. More precisely, we define (LP) to be the linear programming relaxation of (IP) by relaxing the integrality condition $x_e \in \{0,1\}$ to $x_e \geq 0$, and we define the dual (D) of (LP) as :

$$\begin{array}{lll} \text{Max} & \sum_{S \subset V} f(S) y_s & \\ \text{subject to:} & & \\ & \sum_{S : e \in \delta(S)} y_S \leq c_e & e \in E \\ & y_s \geq 0 & S \subset V, S \neq \emptyset \end{array}$$

In the remainder we describe how the greedy algorithm constructs a feasible solution that is at most $(2 - \frac{2}{|A|})$ times the value of a (implicitly constructed) dual solution, with $A = \{v \in V \mid f(\{v\}) = 1\}$. Hence, it follows (see last time's notes) that the algorithm obtains an approximation factor of $2 - \frac{2}{|A|}$. On a sidenote, we mention that the algorithm runs in $O(n^2 \log n)$ time.

```
Input : An undirected graph $G = (V, E)$, edge costs $c_e \geq 0$, and a proper function $f$
Output: A forest $F'$ and a value $LB$
1    $F \leftarrow \emptyset$
2    Comment : Implicitly set $y_S \leftarrow 0$ for all $S \subset V$
3    $LB \leftarrow 0$
4    $\mathcal{C} \leftarrow \{\{v\} : v \in V\}$
5    for all $v \in V$
6        $d(v) \leftarrow 0$
7    while $\exists\, C \in \mathcal{C} : f(C) = 1$
8        Find edge $e = (i, j)$ with $i \in C_p \in \mathcal{C}$, $j \in C_q \in \mathcal{C}$, $C_p \neq C_q$ that minimizes $\epsilon = \frac{c_e - d(i) - d(j)}{f(C_p) + f(C_q)}$
9        $F \leftarrow F \cup \{e\}$
10       for all $v \in C_r \in \mathcal{C}$ do $d(v) \leftarrow d(v) + \epsilon \cdot f(C_r)$
11       Comment : Implicitly set $y_C \leftarrow y_C + \epsilon \cdot f(C_r)$ for all $C \in \mathcal{C}$
12       $LB \leftarrow LB + \epsilon \sum_{C \in \mathcal{C}} f(C)$
13       $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_p \cup C_q\} - \{C_p\} - \{C_q\}$
14   $F' \leftarrow \{e \in F :$ For some connected component $N$ of $(V, F - \{e\})$, $f(N) = 1\}$
```

Figure 5: The algorithm of Goemans and Williamson.

15.2 Description of the Algorithm

The main algorithm is shown in Figure ??. The basic idea behind the algorithm is to build a set of edges F' that corresponds to a feasible solution of (IP). Generally speaking, the algorithm does this as follows:

- Maintain a forest of edges F (F is empty initially);
- Add an edge to F such that two disconnected components of F get merged. Repeat this step until $f(C) = 0$ for all connected components C in F;
- Remove 'unnecessary' edges from F to obtain the final solution F'. An edges e is not necessary if it can be removed from F such that $f(C) = 0$ for all components C of $F - \{e\}$. In words, this means that the removal of e does not affect the validity of the constraints in (IP).

Note that since $f(V) = 0$, the number of iterations is at most $|V| - 1$.

The central part of the algorithm is of course the way we choose an edge at each iteration. This choice is guided by the implicit dual solution we wish to construct: simply stated, we can say that the edge that is selected is such that at each iteration,

1. the dual solution y is maximized, while at the same time
2. y remains a feasible solution to (D).

To be more precise, we define an *active component* to be any component C of F for which $f(C) = 1$. Intuitively, only the dual variables y_C connected with an active component C can increase the dual solution value. Now, the algorithm proceeds greedily. All dual variables y_C associated with active components C are increased uniformly until a dual constraint is met with equality. When this happens, the dual variables cannot be increased further without violating the dual's feasibility, and we select the edge associated with the dual constraint that has become tight.

The next step is to show that the increase ϵ of the dual variables is equal to the formula given in step 8. For this, we first make two observations :

- $d(i) = \sum_{S: i \in S} y_S$ throughout the algorithm for each vertex i. This can easily be shown by induction.
- $\sum_{S: e \in \delta(S)} y_S = d(i) + d(j)$ for all edges $e = (i, j)$ such that i and j are in different components. This follows from the previous observation and from the definition of $\delta(S)$.

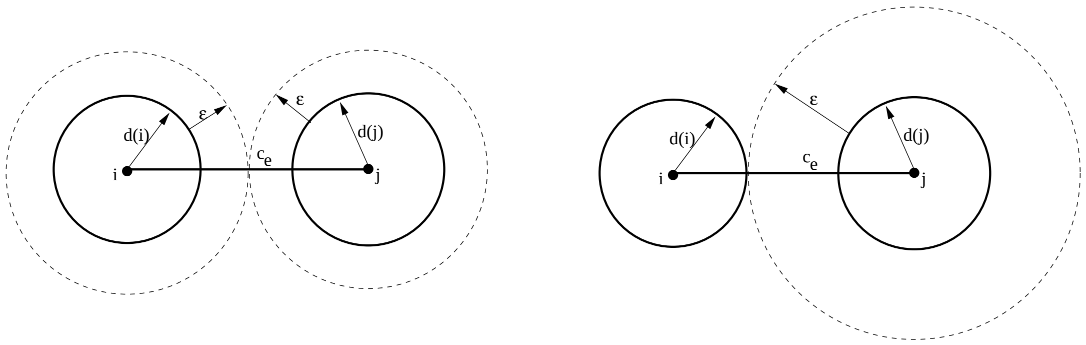

Figure 6: Determination of the dual increase ϵ. The thick radii corresponds to components in F, the dotted radii to a possible increase of the dual variables. In the figure on the left, both components are active. In the figure on the right, only one component is active.

Now, consider any edge $e = (i, j)$ that connects two distinct components C_p and C_q in F. If we increase the dual variables y_C by ϵ, $\sum_{e \in \delta(S)} y_S$ will increase by $\epsilon \cdot f(C_p) + \epsilon \cdot f(C_q)$. Hence, we can reformulate the dual constraint associated with e as

$$d(i) + d(j) + \epsilon \cdot f(C_p) + \epsilon \cdot f(C_q) \leq c_e$$

Then, it follows that the largest possible increase of ϵ that does not violate the feasibility of the dual is exactly the one given in step 8. The relation between $d(i), d(j), f(C_p, f(C_q)$ and ϵ is illustrated in Figure **??**.

We also mention that the algorithm enforces the complementary slackness conditions of the primal (see notes lecture 24), that is, for all $e \in E$ we have $e \in F \Rightarrow c_e = \sum_{S: e \in \delta(S)} y_S$. This immediately follows from the preceding discussion, since we only add e to F if the corresponding dual constraint is met with equality.

15.3 Analysis

At this point, we still need to prove three things : the feasibility of the final primal and dual solutions, and the actual approximation bound. We first prove the feasibility of the final dual solution.

Theorem 15.1 *The dual solution y constructed by the algorithm is feasible for (D).*

Proof:
By induction. It is evident that y is feasible initially, since all the dual variables y_S are 0. By the preceding discussion, we have that the dual constraint associated with an edge $e = (i, j)$ holds throughout the algorithm if i and j are in different components. In case i and j are in the same component, the left hand side $\sum_{S: e \in \delta(S)} y_S$ of the dual constraint does not increase, and therefore the constraint continues to hold. □

Now, we show that the algorithm produces a feasible solution to the integer program (IP). The idea behind the proof is to show that if $f(C) = 0$ for all connected components C in F', the solution corresponding to F' is feasible. For this to apply however, we first need to show that $f(C) = 0$ for all connected components C in F'. This is captured in the following lemma's.

Lemma 15.2 *If $f(S) = 0$ and $f(B) = 0$ for some $B \subseteq S$, then $f(S - B) = 0$.*

Proof:
By the symmetry property of f, $f(V - S) = f(S) = 0$. By the disjointness property, $f((V - S) \cup B) = 0$. Then, by the symmetry property again, we have $f(S - B) = f((V - S) \cup B) = 0$ □

Lemma 15.3 *For each connected component N of F', $f(N) = 0$.*

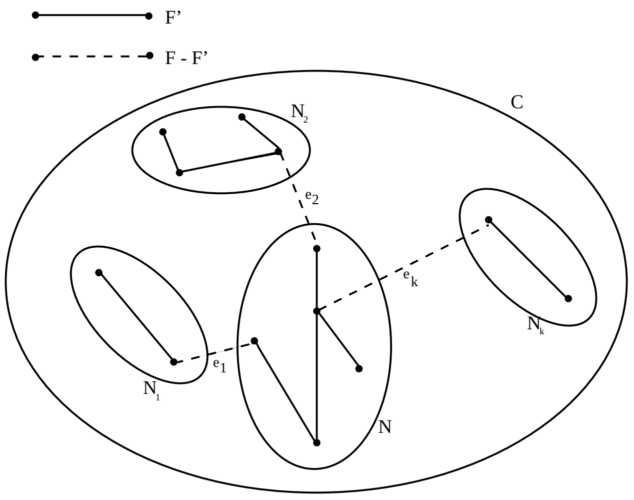

Figure 7: Illustration of Lemma 1.3.

Proof:
By construction of F', we have that $N \subseteq S$ for some component C of F. Let $e_1, \ldots, e_k$ be edges of F that have exactly one endpoint in N. These are the edges in C that were removed in the final step of the algorithm. Let N_i and $C - N_i$ be the two components that were generated by removing e_i from C, with $N \subseteq C - N_i$(see Figure **??**). Since e_i was removed, we have that $f(N_i) = 0$. Also, the sets $N, N_1, \ldots, N_k$ form a partition of C. So, by the disjointness property of f we then have $f(C - N) = f(\bigcup_{i=1}^{k} N_i) = 0$. Because $f(C) = 0$, the previous lemma then implies that $f(N) = 0$. □

Theorem 15.4 *The solution associated with F' is a feasible solution to (IP).*

Proof:
By contradiction. Suppose that some constraint is violated, i.e., $\sum_{e \in \delta(S)} x_e = 0$ for some S such that $f(S) = 1$. This means that there is no edge in F' that has exactly one endpoint in S, so for all connected components N_i in F', either $N \cap S = \emptyset$ or $N \cap S = N$. So, $S = N_{i_1} \cup \ldots \cup N_{i_k}$ for some $i_1, \ldots, i_k$ However, by the previous lemma $f(N) = 0$, and then by the disjointness property $f(S) = 0$ too. But this contradicts our assumption, which completes the proof. □

Finally, we show that the algorithm obtains an approximation factor of $2 - \frac{2}{|A|}$. To do this, we only need to prove that the value of the primal solution associated with F' is at most $2 - \frac{2}{|A|}$ times the value of the implicitly constructed dual solution y. This is done in the following theorem.

Theorem 15.5 *The algorithm produces a set of edges F' and a dual solution y such that*

$$\sum_{e \in F'} c_e \leq (2 - \frac{2}{|A|}) \sum_{S \subset V} y_s$$

Proof:
We have that

$$\begin{aligned} \textstyle\sum_{e \in F'} c_e &= \textstyle\sum_{e \in F'} \sum_{S : e \in \delta(S)} y_s && \text{by the primal complementary slackness conditions} \\ &= \textstyle\sum_{S \subset V} y_s \cdot |F' \cap \delta(S)| && \text{by exchanging the summations} \end{aligned}$$

Now, we show by induction on the number of iterations that

$$\sum_{S \subset V} y_s \cdot |F' \cap \delta(S)| \leq (2 - \frac{2}{|A|}) \sum_{S \subset V} y_s$$

It is easy to see that this inequality holds initially, because then all y_s are 0. The next step is then to prove that the inequality remains valid from one iteration to another. This is done by showing that at each iteration, the increase in the left hand side of the equation is at most the increase in the right hand side. If we let $\mathbf{C}$ be the set of components at the beginning of the iteration and $\mathbf{C}^1 = \{C \in \mathbf{C}: f(C) = 1\}$ the set of active components, this amounts to proving

$$\sum_{C \in \mathbf{C}^1} \epsilon \cdot |F' \cap \delta(C)| \leq (2 - \frac{2}{|A|})\epsilon \cdot |\mathbf{C}^1|$$

To prove this, we first observe that $F' \cap \delta(C)$ consists of those edges in F' that have exactly one endpoint in C, so if we looked at a component C as being a vertex, $|F' \cap \delta(C)|$ would be its degree. The inequality then states that the average degree of the active components is at most $(2 - \frac{2}{|A|})$ (divide both sides by ϵ and $|\mathbf{C}^1|$).

To see this, we construct a graph H whose vertices are the components $C \in \mathbf{C}$. Its edges are $e \in F' \cap \delta(C)$ for all C, that is, those edges of F' that connect components in $\mathbf{C}$. Isolated vertices that correspond to inactive components are removed. Note that since F' is a forest, H is a forest too.

The key observation to make about the graph H is that it has no leaf that corresponds to an inactive component (so all inactive components have a degree of at least 2). To see this, we suppose otherwise, and let v be a leaf, C_v its associated inactive component, and e the edge incident on v. Now, by removing e, the component N in H that contains v is partioned into components $\{v\}$ and $N - \{v\}$. Let $C_{N-\{v\}}$ be the union of the components assiocated with $N - \{v\}$. We know that $f(C_v) = 0$ and since there are no edges leaving $N - \{v\}$, we also have $f(C_{N-\{v\}}) = 0$. But this means that e was removed at final step of the algorithm, so we have a contradiction.

We now finalize the proof. Let d_v be the degree of a vertex v in H, and let N_a, N_i be the sets of vertices in H corresponding to active resp. inactive components. Given the observations above, we then have

$$\sum_{v \in N_a} d_v = \sum_{v \in N_a \cup N_i} d_v - \sum_{v \in N_i} d_v \leq 2(|N_a| + |N_i| - 1) - 2|N_i| = 2|N_a| - 2$$

The inequality holds since in any forest, the number of edges is less than the number of vertices. Multiplying by ϵ and substituting N_a by $\mathbf{C}^1$, we then get

$$\sum_{C \in \mathbf{C}^1} \epsilon \cdot |F' \cap \delta(C)| \leq 2\epsilon(|\mathbf{C}^1| - 1) \leq (2 - \frac{2}{|A|})\epsilon \cdot |\mathbf{C}^1|$$

because the number of active components at any iteration will be at most $|A|$. $\square$