CMPUT 675: Approximation Algorithms **Fall 2014**

Lecture 13 (Oct 1): MULTICUT (Part 1)

Lecturer: Zachary Friggstad *Scribe: Chris Martin*

13.1 Multicut

The MULTICUT problem given as follows. We are given an *undirected* graph $G = (V, E)$ with edge costs $c_e \geq 0$, $e \in E$, and k pairs of nodes $(s_1, t_1), \cdots (s_k, t_k)$ where $s_i \neq t_i$ for all i. The goal is to find the cheapest set of edges $F \subseteq E$ such that all (s_i, t_i) pairs are disconnected in the graph $(V, E - F)$ (*i.e.* there is no $s_i - t_i$ path in the new graph). The problem can be solved in polynomial time when $k = 1$. For example, compute a maximum $s_1 - t_1$ flow with edge capacities c_e and find the set S of all vertices reachable from s_1 in the residual graph; the set $\delta(S)$ is a minimum-cost $s_1 - t_1$ cut. For larger values of k, the problem is NP-hard.

This problem can be formulated as an integer program as follows. Let the binary variables $x_e \in \{0, 1\}$ denote whether edges $e \in E$ are cut, and let $\mathcal{P}_i$ denote the set of all $s_i - t_i$ paths. Note that the size of $\mathcal{P}_i$ can be exponential in the worst case. The integer program is:

$$\text{minimize:} \quad \sum_{e \in E} c_e \cdot x_e \qquad \textbf{(MC-IP)}$$

$$\text{subject to:} \quad \sum_{e \in P} x_e \geq 1, \text{ for all } 1 \leq i \leq k, P \in \mathcal{P}_i \qquad (13.1)$$

$$x_e \in \{0, 1\}, \text{ for all } e \in E. \qquad (13.2)$$

Constraint 13.1 ensures that each $s_i - t_i$ path has at least one edge cut, while the minimization objective function ensures that the minimum cost cut is obtained. The LP relaxation of this problem simply allows the variables x_e to be within the range [0,1]:

$$\text{minimize:} \quad \sum_{e \in E} c_e \cdot x_e \qquad \textbf{(MC-LP)}$$

$$\text{subject to:} \quad \sum_{e \in P} x_e \geq 1, \text{ for all } 1 \leq i \leq k, P \in \mathcal{P}_i \qquad (13.3)$$

$$x_e \in [0, 1], \text{ for all } e \in E. \qquad (13.4)$$

Since the **MC-LP** problem has exponentially many constraints, we cannot just use any polynomial-time LP solver to efficiently compute an optimum solution. Instead, we must use a different approach.

13.2 Separation Oracles

Definition 1 *Let $\mathcal{P}$ be the set of feasible solutions to an LP on n variables. A **separation oracle** for $\mathcal{P}$ is an algorithm that takes as input a potential LP solution $\bar{x} \in \mathbb{Q}^n$ and in time poly(n, bit complexity of $\bar{x}$) either:*

1. *Correctly determines $\bar{x} \in \mathcal{P}$, or*
2. *Produces a constraint separating $\bar{x}$ from $\mathcal{P}$ (*i.e. *a constraint that $\bar{x}$ violates).*

For example, a separation oracle for **MC-LP** is given by Algorithm 1.

Algorithm 1 Oracle for **MC-LP**

Input: Undirected graph $G = (V, E)$, LP constraints $\mathcal{P}$, and potential LP solution $\bar{x}$.
Output: Accepts or provides a constraint that $\bar{x}$ violates.

```
Check x̄ ∈ [0,1]^E
G' ← the graph G with edge lengths x̄
for each 1 ≤ i ≤ k do
    if the shortest s_i − t_i path P has length < 1 then
        return The constraint corresponding to P
    end if
end for
return ACCEPT
```

This algorithm is correct: if the shortest $s_i - t_i$ path has length ≥ 1 then all constraints corresponding to paths from s_i to t_i must be satisfied since they are all longer. If the shortest $s_i - t_i$ path has length < 1 then such a shortest path P corresponds to a violated constraint.

It turns out that having such a separation oracle is all that is needed for solving a linear program in polynomial time.

Theorem 1 *A linear program* $\min\{c^T x : A \cdot x \geq b, x \geq 0\}$ *with maximum bit complexity* Δ *and* $x \in \mathbb{Q}^n$ *can be solved in time poly*(n, Δ) *if the feasible solutions admit a separation oracle. Furthermore, if the linear program has an optimal solution then an extreme point optimum solution is returned.*

The algorithm is known as the *Ellipsoid method*. A full proof of this theorem, including the bit complexity analysis, can be found in the book *Combinatorial Optimization: Theory and Algorithms* by Korte and Vygen [KV00].

Note that the running time does not depend on the number of constraints. Theorem 1 says that it is enough to efficiently separate over the constraints in order to solve a linear program in polynomial time. Therefore, the **MC-LP** problem can be solved in polynomial time using the oracle given in Algorithm 1.

13.3 Solving Multicut Using MC-LP (Part 1)

The algorithm for converting a solution to **MC-LP** into a solution for MULTICUT was given and proven correct in the lecture. However, the proofs of the integrality gap in Theorem 2 and the crucial Lemma 2 were delayed until the next lecture, so they will be assumed true for the time being.

Assume that an optimal solution $\mathbf{x}^*$ to the **MC-LP** problem has been computed. For vertices $u, v \in V$, let $d(u, v)$ be the shortest-path distance from u to v in G where each edge $e \in E$ has length x_e^*. Note that d satisfies the triangle inequality since it is a shortest path metric.

Definition 2 *Consider a subgraph G' of G, a vertex v in G' and a* radius $r \geq 0$. *The* ***ball*** $B_{G'}(v, r)$ *is the set of all vertices u of G' such that $d(u, v) \leq r$.*

Note that the balls $B_{G'}(v,r)$ are defined with respect to the distances $d(u,v)$ in the original graph G. This is ok.

We note the following properties of these balls.

Lemma 1 *For any subgraph G' of G, any $1 \le i \le k$ such that s_i, t_i both lie in G', any vertex v of G', and any a radius $r \in [0, 1/2)$, $|B_{G'}(v,r) \cap \{s_i, t_i\}| \le 1$.*

Proof. If both $s_i, t_i \in B_{G'}(v,r)$ then by the triangle inequality $d(s_i,t_i) \le d(s_i,v) + d(v,t_i) \le r + r < 1$, again contradicting Constraints (13.3). ■

Definition 3 *Let G' be a subgraph of G, v be a vertex of G', and, $r \ge 0$, and $\mathbf{x}^*$ be the optimal solution to* **MC-LP**. *The* ***volume*** *$V_{G'}(v,r)$ is given by:*

$$V_{G'}(v,r) = \frac{OPT_{LP}}{k} + \sum_{\substack{e=(u,w) \\ u,w \in B_{G'}(v,r)}} c_e \cdot x^*_e + \sum_{\substack{e=(u,w) \in G' \\ u \in B_{G'}(v,r) \\ w \notin B_{G'}(v,r)}} c_e \cdot (r - d(v,u)).$$

The term OPT_{LP}/k might seem a bit arbitrary, but it is carefully chosen to make the analysis of the integrality gap (next lecture) work correctly. It will be discussed in a bit more detail then. The second term is the contribution to the objective function of all edges completely contained in the ball $B_{G'}(v,r)$. For an edge $e = (u,w)$ contributing to the third term, the value $r - d(v,u) \in [0, x^*_e)$ (so if r is increased to $r' := d(u,v) + x^*_e$ then surely $w \in B'_G(v,r')$).

Recall the standard notation that for a subset of vertices $S \subseteq V$ of a graph G, $\delta_G(S)$ denotes the edges in G that join a vertex in S to a vertex in $V - S$. Using these definitions, the process to convert a **MC-LP** solution to a MULTICUT solution is given as Algorithm 2.

Algorithm 2 Conversion to MULTICUT

Input: Undirected graph $G = (V, E)$.
Output: The cut F.

1: $G' \leftarrow G$
2: $\mathbf{x}^* \leftarrow$ the optimum solution to **MC-LP**
3: $F \leftarrow \emptyset$
4: **while** some (s_i, t_i) pair is connected in G' **do**
5: Let $r \in [0, \frac{1}{2})$ be as promised by Lemma 2 (below).
6: $F \leftarrow F \cup \delta_{G'}(B_{G'}(s_i, r))$
7: Remove the vertices $B_{G'}(s_i, r)$ and their incident incident edges from G'.
8: **end while**
9: **return** F

Claim 1 *The result F computed by Algorithm 2 separates all $s_i - t_i$ pairs.*

Proof. First, note that for any subgraph G' considered in any iteration with vertex set, say, $\delta_G(S) \subseteq F$. This is easy to prove by induction.

Consider any pair (s_j, t_j). Since the algorithm terminates only when there is no (s_i, t_i) pair left in the subgraph G' then there was some iteration where at least one of the two nodes was in the ball that was removed. Consider the earliest iteration where one of these nodes was in such a ball and suppose this vertex is s_j (without loss of

generality). Say that the pair (s_i, t_i) is the pair considered in this iteration (it may or may not be that $i = j$) and now say that G' is subgraph of G in this iteration just before the ball $B_{G'}(s_i, r)$ is removed. That is, we are assuming $s_j \in B_{G'}(s_i, r)$. By Lemma 1, we cannot have $t_j \in B_{G'}(s_i, r)$.

Now consider any $s_j - t_j$ path P in the original graph. If P is not contained entirely in the graph G' just before $B_{G'}(s_i, r)$ is removed then the observation in the first sentence shows that P already contained some edge in F. If P is contained entirely in this subgraph G', then because it starts in $B_{G'}(s_i, r)$ and ends at $t_j \notin B_{G'}(s_i, r)$ it must use some edge in $\delta(B_{G'}(s_i, r))$. This edge is added to F.

In either case, P contains an edge from F. Since this is true for all $s_j - t_j$ paths, the pair (s_j, t_j) is disconnected in the graph $(V, E - F)$. ■

The remaining Theorem and Lemma will be stated here but not proven until the next lecture.

Lemma 2 *Consider an iteration of the algorithm, let G' be the current graph in that iteration, and let v be any vertex of G'. There exists some $r \in [0, 1/2)$ such that $c(\delta_{G'}(B_{G'}(v, r))) \leq 2 \cdot \ln(k+1) \cdot V_{G'}(v, r)$.*

Theorem 2 *The integrality gap of* **MC-LP** *is at most $4 \cdot \ln(k+1)$.*

References

KV00 B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, Springer-Verlag, Berlin, 2000.