Approximation Algorithms II

Review of Part I

complexity classes:

- P : class of problems solvable in poly time ($O(n^k)$)
- NP : class of problems verifiable in poly time

NP-Hard → really hard problems

Approx. Algorithms:

- provide sup-optimal solutions
- solutions found in poly time despite problem being NP-Hard
- can be used to find faster poly. time solutions to problems that already have poly time solutions

For a given approx. algorithm:

Denote OPT = optimal solution to problem

Denote C = solution found by approx. algo

Let's say $C \leq (2) \cdot OPT$ → represent bounds on how much worst C is than OPT

or $C \leq (1.5) \cdot OPT$

$P(n) = \max\left(\frac{OPT}{C}, \frac{C}{OPT}\right)$ = approximation ratio.

PTAS ⇒ Polynomial Time Approximation Scheme

given ε as input, we can have a $(1+\varepsilon)$ - approximation algorithm to given problem.

Some Graph Theory Concepts

Given $G=(V,E)$

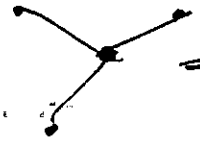
Cut: Partition of vertices into two disjoint subsets.

Cut-Set (referred to as cut too): Set of edges whose end points are in diff. subsets of the partition.

S-T cut: Cut in which S and T in different subsets

Minimum cut: Cut with least weight of possible cut-set in a weighted graph $G(V,E)$

Terminal vertex: Vertex of degree 1.

Star:  ⇒ is a $K_{1,3}$ star.

$K_{1,3}$: center ↙, number of terminals ↓

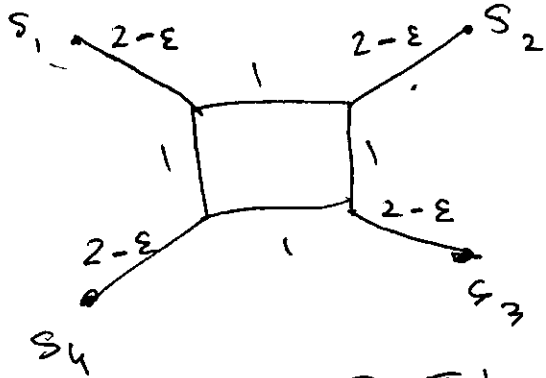
Multiway Cut Problem

$G=(V,E)$ { connected, undirected, weighted

Problem: Given set of terminals $S=\{s_1, s_2, \ldots, s_k\} \subseteq V$,

a multiway cut is a set of edges whose removal disconnects terminals from each other. This problem asks for minimum weight of such a set.

Example:



$S=\{s_1, s_2, s_3, s_4\}$

Possible multiway cuts: (1) Take out all 1 edges

(2) Take out 3 $(2-\varepsilon)$ edges $\Rightarrow$ (minimum)

⋮

many

Def. **Isolating Cut**

Isolating cut for s_i is set of edges whose removal disconnects s_i from rest of terminals.

Approx. Algorithm:

(1) For each $i=1, \ldots, k$, compute minimum weight isolating cut for s_i, say C_i.

(2) Discard heaviest of these cuts, and output union of the rest, say C.

Each computation can be accomplished by combining all terminals other than s_i into a single node, and running the max-flow algorithm once.

Theorem: Approx. Algo. achieves guarantee of $2 - 2/k$.

Proof: Let A^* = optimal multiway cut in G. Removal of A^* creates k connected components, each with one terminal. Let A_i = cut separating a component containing s_i from rest of the graph, then

$$A^* = \bigcup_{i=1}^{k} A_i .$$

Since each edge in A^* incident at two of these components, ~~each edge of A^* incident at two of these components~~ each edge will be in two cuts of A_i, and hence

$$\sum_{i=1}^{k} w(A_i) = 2\, w(A^*).$$

A_i is an isolating cut for s_i, and C_i minimum weight isolating cut for s_i, $w(C_i) \le w(A_i)$. Further, C is obtained by discarding heaviest of the cuts C_i,

$$w(C) \le \left(1 - \frac{1}{k}\right) \sum_{i=1}^{k} w(C_i) \le \left(1 - \frac{1}{k}\right) \sum_{i=1}^{k} w(A_i) = 2\left(1 - \frac{1}{k}\right) w(A^*).$$

Minimum K-Cut Problem

k-Cut: Set of edges whose removal leaves k connected components.

Minimum k-Cut Problem: asks for minimum weight k-cut for $G = (V, E)$.

For $k \geq 3$, NP-Hard

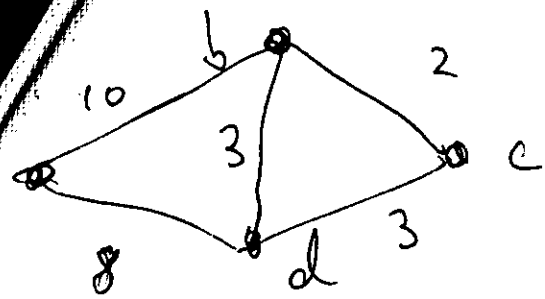
$k = 2 \Rightarrow$ Same case as s-t cut.

Natural Algorithm:

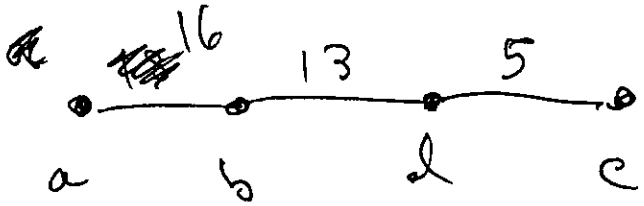
Start with G.
Compute minimum cut in each connected component - take out smallest one
Repeat until you have k connected components.
Achieves $2 - 2/k$ approx., but proof complicated.

Gomory-Hu tree representation of minimum cuts

- Weighted tree that represents minimum s-t cuts for all s-t pairs in the graph. It can be constructed in $|V| - 1$ minimum cut computations.
- Cut defined in graph G by partition $(S, \bar{S})$ is cut associated with e in G.
- T is a Gomory-Hu tree for G if
  1. for each pair $(u, v) \in V$, weight of minimum u-v cut in G is same as in T.
  2. For each edge $e \in T$, $w'(e)$ is weight of cut associated with e in G.



Graph G



Gomory-Hu Tree T

Approx. Algorithm

1. Compute Gomory-Hu tree T for G.
2. Output union of lightest k-1 cuts of the n-1 cuts associated with edges of T in G; let C be their Union.

if S = Union of cuts in G associated with l edges of T, then removal of S from G leaves a graph with at least l+1 components.

Theorem: Approx. algorithm achieves factor of $2 - 2/k$.

Proof. A = optimal k-cut in G, A is union of k cuts $A_1, \ldots, A_k$.
consisting of connected components $V_1 \ldots V_k$.

A_i = cut separating V_i from rest of graph.

Each edge of A lies in 2 cuts,

$$\sum_{i=1}^{k} w(A_i) = 2\, w(A).$$

W.L.O.G. $\Rightarrow$ A_k is heaviest of these cuts.

Let B be set of edges of T that connect across two of the sets $V_1, V_2, \ldots, V_k$. Consider graph on vertex set V and edge set B, and shrink $V_1, V_2, \ldots, V_k$ to a single vertex.

Shrunk graph is connected since T is connected.

Throw away edges till tree remains, and let remaining $k-1$ edges denote the required $k-1$ cuts. (B')

Root the tree at V_k.

Let edge $(u,v) \in B'$ correspond to set V_i in this manner. Weight of minimum u-v cut in G is $w'(u,v)$. Since A_i is a u-v cut in G,

$$w(A_i) \geq w'(u,v).$$

Each cut among $A_1, A_2, \ldots A_{k-1}$ is at least as heavy as cut defined in G by corresponding edge of B'.

$$w(C) \geq \sum_{e \in B'} w'(e) \leq \sum_{i=1}^{k-1} w(A_i) \leq \left(1-\frac{1}{k}\right) \sum_{i=1}^{k} w(A_i) = 2\left(1-\frac{1}{k}\right) w(M)$$

K- center Problem

Given set of cities, pick k cities for locating warehouses so as to minimize max. distance of a city from its closest warehouse.

Formally:

K center Problem

Let $G=(V,E)$ be complete undirected graph with edge costs satisfying triangle inequality, and k be a positive integer. For any set $S \subseteq V$ & vertex $v \in V$, define connect (v,S) to be the cost of cheapest edge from v to a vertex in S. Find set $S \subseteq V$ with $|S| = k$ so as to minimize $\max_v \{ \text{connect}(v,S) \}$.

Parametric Pruning → choose parameter t, instance I.
Instance I is pruned by removing parts that will not be used in any solution of cost $\leq t$.

Sort edges of G in non decreasing order:

$$\text{cost}(e_1) \leq \text{cost}(e_2) \ldots \text{cost}(e_m).$$

let $G_i = (V, E_i)$ where $E_i = \{e_1, e_2, \ldots e_i\}$.

Dominating Set: In undirected graph $H=(V,F)$, is a subset $S \subseteq V$ such that every vertex in $V-S$ is adjacent to a vertex in S.

Dom(H) Size of minimum cardinality dominating set in H.

k-center problem $\Longleftrightarrow$ finding smallest index i such that G_i has a dominating set of size atmost k, cost(e_i) is cost of such k-centers

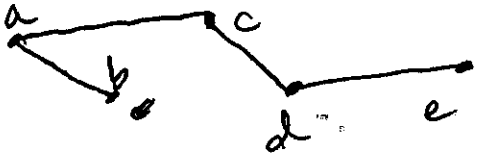
$\therefore$ G contains k stars spanning all vertices.

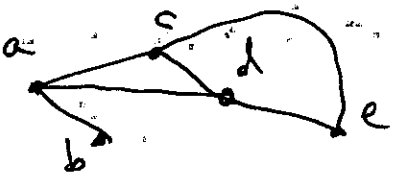
Star: $K_{1,p}$ for $p \geq 1$

example $\Rightarrow$  $= K_{1,3}$.

Square of graph H: Graph containing edge (u,v) whenever H has a path of length at most two between u and v, $u \neq v$.

Denote H^2.





Example: H H^2

Lemma: Given graph H, let I be independent set in H^2.

Then $|I| \leq dom(H)$.

Proof: Let D be a minimum dominating set in H.

H contains $|D|$ stars spanning all vertices, and each of these will be a clique in H^2 spanning and vertices. I can pick at most one vertex from each clique, and the lemma follows.

Approx. Algorithm:

1. Construct $G_1^2, \ldots G_m^2$
2. Compute maximal independent set M_i in each graph G_i^2
3. Find smallest index i such that $|M_i| < k$, say j.
4. Return M_j.

Theorem: Approx. algorithm achieves factor of 2 for k-center problem

Proof.

Maximal independent set, I, is also a dominating set. If v is not dominated by I, then it should be in I and we have contradiction regarding I's cardinality.

Now, there exists stars in G_j^2 centered on vertices M_j, covering all vertices. Each edge used in constructing these stars has cost at most $2 \cdot cost(e_j)$.

Lemma For j as defined in algorithm, $cost(e_j) < OPT$.

Proof: For every $i \leq j$, $|M_i| > k$, and therefore $dom(e_i) > k$, and so $i^* \geq i$. Hence, $j < i^*$.

Weighted k-center Problem

Problem. In addition to cost function on edges, we have a weight function on vertices. Find $S \subseteq V$ of total weight at most W, where $w: V \to R^+$ and bound $W \in R^+$, while minimizing

$$\max_{v \in V} \left\{ \min_{u \in S} \{ cost(u,v) \} \right\}.$$

wdom(G) = weight of minimum weight dominating set in G.
We need to find smallest index i s.t. wdom(G_i) $\leq W$.
If i^* is this index, then cost of OPT solution is $OPT = cost(e_{i^*})$

Given vertex weighted graph H, let I be an independent set in H^2. For each $u \in I$, let $s(u)$ denote lightest neighbor of u in H, where u is also considered a neighbor of itself. (neighbor picked in H and not H^2).

Let $S = \{ s(u) \mid u \in I \}$

Lemma $w(S) \leq$ wdom(H)

Proof: D = minimum weight dominating set of H.
There exist disjoint stars in H centered on vertices of D covering all vertices. Each star becomes a clique in H^2, and I can pick at most one vertex from each. Then, each vertex in I has center of corresponding star available as a neighbor in H.
Hence, $w(S) \leq$ wdom(H).

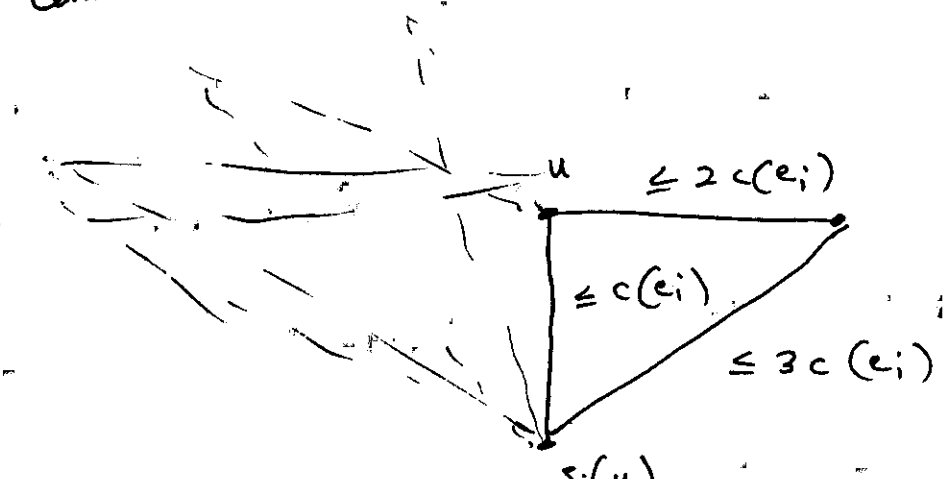
Approx. Algorithm

1. Construct $G_1^2, \ldots, G_n^2$
2. Compute max. ind. set, M_i, in each graph G_i^2
3. Compute $S_i := \{s_i(u) \mid u \in M_i\}$
4. Find minimum index j such that $w(S_i) \leq W$, say j.
5. Return S_j.

where $s_i(u)$ denotes lightest neighbor of u in G_i, and u also considered a neighbor of itself.

Theorem Approx. Algorithm achieves factor of 3 in weighted k-center problem.

Pf. $cost(e_j)$ is a lower bound on OPT. Since M_j is a dominating set in G_j^2, we can cover V with stars of G_j^2 centered in vertices of M_j.



Each star center is adjacent to a vertex in S_j, using an edge of cost at most $cost(e_j)$. Move each of the centers to the adjacent vertex in S_j and ~~reduce~~ redefine the stars. By triangle inequality, largest edge cost used in constructing the final stars is at most $3 \cdot cost(e_j)$.

Approx. Algorithms

Pros

- worst case ratio robust
- explains why problems vary in difficulty
- analysis reveals difficult vs. easy cases
- can get practical heuristics
- sophisticated / beautiful ideas

Cons

- worst-case oriented; can ignore algorithms that work well on average
- limited to cleanly stated problems
- framework does not apply to decision problems.

Broadly,

- not limited to NP-Hard problems
- faster polynomial time algorithms
- limited space (external memory algorithms)
- limited access to data (streaming algorithms)