CS 255, Spring 2014, SJSU

Approximation Algorithms and Local Search

Fernando Lobo

Approximation Algorithms

- Often applied to solve optimization problems (want to maximize or minimize some objective.)
- Most NP-Complete problems have a corresponding optimization version
  - Find a maximum clique of an undirected graph.
  - Find a minimum vertex cover of an undirected graph.
  - ...

Approximation Algorithms

- Many practical problems are NP-Complete. What do do?
- If instance is small use a brute-force approach.
- Otherwise use an approximation algorithm.
  - no guarantee of getting optimal solution
  - in practice we are happy with a near-optimal solution.

Approximation ratio

- Optimization problem where each potential solution has a positive cost.
- Suppose optimal solution has cost C^*
- An algorithm has an *approximation ratio* $\rho(n)$ if for any input of size n, the cost C of the solution produced by the algorithm has a cost within a factor of $\rho(n)$ of the optimal cost C^*
  - For maximization problems: $C^*/C \leq \rho(n)$
  - For minimization problems: $C/C^* \leq \rho(n)$
- Approximation ratio gives us a solution quality guarantee, even if the optimal solution is unknown.

Example 1: Vertex Cover

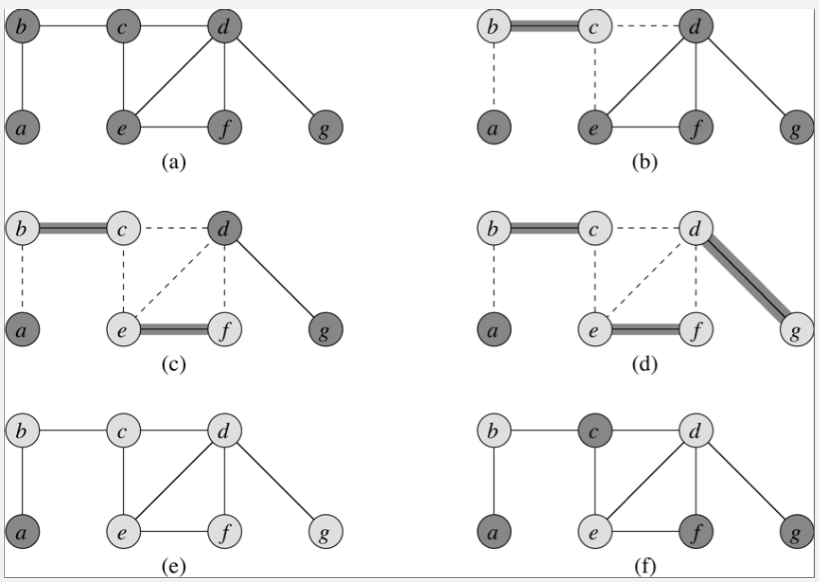
- Given an undirected graph $G = (V, E)$, we say $S \subseteq V$ is a vertex cover if and only if for every edge $(u, v) \in E$, at least one of the ends (u or v) belong to S.
- Optimization problems: Given an undirected graph $G = (V, E)$, find a vertex cover of maximum size.

An approximation algorithm for Vertex Cover

Approx-Vertex-Cover(G)

```
S = ∅
E' = G.E
while E' ≠ ∅
    let (u, v) be an arbitrary edge of E'
    S = S ∪ {u, v}
    remove from E' every edge incident on either u or v
return S
```


Operation of Approx-Vertex-Cover



(Figure taken from CLRS textbook)

Correctness

- Approx-Vertex-Cover returns a vertex cover.
- The algorithm loops until every edge of the input graph has been covered by some vertex in S.

APPROX-VERTEX-COVER has $\rho(n) = 2$

- Let S^* be an optimal vertex cover.
- Let A be the set of edges processed on line 4 of the algorithm.
- S^* must include at least one endpoint of each edge in A (otherwise it wouldn't be a vertex cover).
- No two edges in A share an endpoint (because line 6 of the algorithm deletes all other incident edges on its endpoints).
  - $|S^*| \geq |A|$
  - On the other hand, $S = 2|A|$
  - This implies $|S| \leq 2|S^*|$

Example 2: Traveling Salesman Problem (TSP)

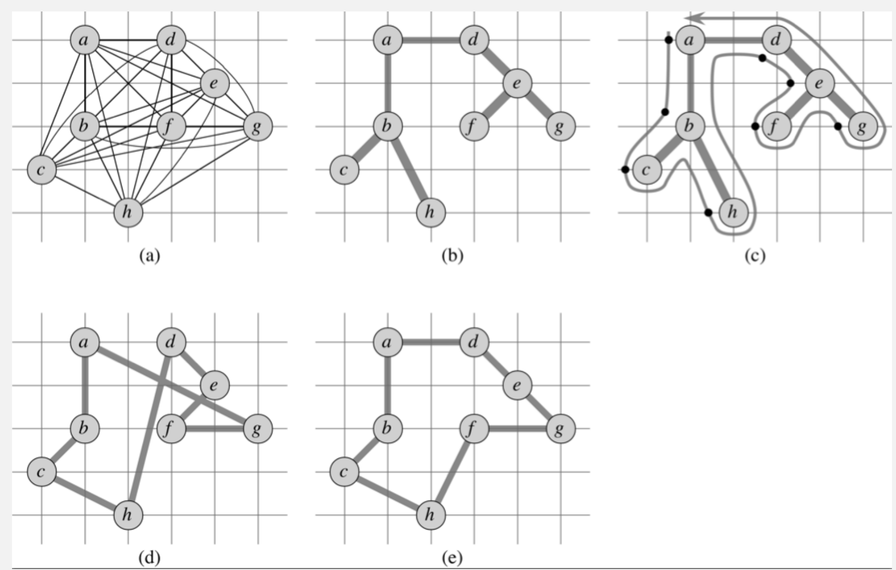
- Given a complete undirected graph $G = (V, E)$ that has a non-negative cost $c(u, v)$ associated with every edge $(u, v) \in E$, find an hamiltonian cycle of G (a tour that visits each vertex of the graph exactly once) with minimum cost.
- We shall assume the cost function c obeys to the triangle inequality:
  - $c(u, w) \leq c(u, v) + c(v, w)$

An approximation algorithm for TSP

APPROX-TSP-TOUR(G, c)

select a vertex $r \in G.V$ to be a root vertex
call MST-PRIM(G, c, r) to produce a minimum spanning tree T
let H be a list of vertices, ordered according to when they are first visited in a preorder tree walk of T
return the hamiltonian cycle H

Operation of APPROX-TSP-TOUR



(Figure taken from CLRS textbook)

Operation of Approx-TSP-Tour (see Figure in previous slide)

- (a) complete graph
- (b) MST obtained using vertex a as root
- (c) A walk of T starting in a
  Full walk: $a, b, c, b, h, b, a, d, e, f, e, g, e, d, a$
  Preorder walk: a, b, c, h, d, e, f, g
- (d) tour H returned by Approx-TSP-Tour
- (e) an optimal tour H^*

Approx-TSP-Tour has $\rho(n) = 2$ (if c obeys to the triangle inequality)

- Let H^* be an optimal tour.
- Deleting an edge from H^* gives a spanning tree of G
- Since T is a MST of G, we know that $c(T) \leq c(H^*)$
- A full walk of T traverses each edge of the tree exactly twice $\Longrightarrow$ its cost is $2\ c(T)$
- But $c(H) \leq c(\text{full walk})$ because of triangle inequality.
- This implies, $c(H) \leq 2\ c(T)$

Local search

- Rely on the notion of neighborhood
- Start with some initial solution X
- Obtain a solution X' in the neighborhood of X
- If X' has a better solution quality than X, then replace X by X'.
- Keep going until no further improvement is possible.
- X is the final solution.

Example: Vertex Cover

- Given a graph, a potential solution X is any subset of vertices of the graph.
- A neighbor of a solution X is a solution X' that differs from X by adding or removing a vertex. (Note: we could define a different neighborhood.)
- Need to have a cost function that measures the quality of a solution.

A cost function for Vertex Cover (Khuri and Bäck, 1994)

- Let $G = (V, E)$ be an undirected graph.
- Any subset of vertices $S \subseteq V$ is a candidate vertex cover.
- A subset S over n elements can be represented by an n-bit binary string $X = x_1 x_2 \ldots x_n$, where
  - $x_i = 1$ if the i^{th} element belongs to S
  - $x_i = 0$ otherwise.

A cost function for Vertex Cover (Khuri and Bäck, 1994)

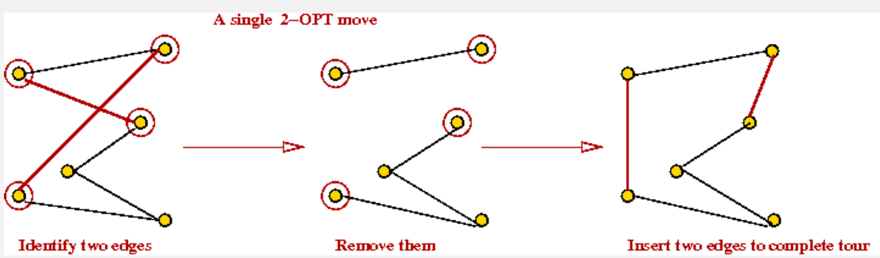
Khuri and Bäck (1994) defined the following function,

$$f(X) = \sum_{i=1}^{n} \left(x_i + n(1 - x_i) \sum_{j=1}^{n} (1 - x_j) e_{ij}\right)$$

- term $\sum_{i=1}^{n} x_i$ gives size of potential cover
- term $n \sum_{i=1}^{n} \sum_{j=1}^{n} (1 - x_i)(1 - x_j) e_{ij}$ penalizes sets that are not covers by adding a penalty of size n for each edge e_{ij} that violates the cover condition (i.e., when $i \notin S$ and $i \notin S$)
  - Any feasible solutions (a vertex cover) has a better cost than an infeasible solution.
  - Penalty is graded: the further away from being feasible, the higher the penalty.

Example: TSP

- 2-opt induces a neighborhood for TSP.
- 2-opt: given a tour, remove 2 edges and reconnect the two resulting tours in another way obtained another tour.



From 2-opt to k-opt

- 2-opt can be generalized to k-opt
- k-opt: given a tour, remove k edges and reconnect the resulting tours in all possible ways, and keep the one with minimum cost.
- With larger k the neighborhood gets larger
  - easier to escape local optima
  - but algorithm is more time consuming, size of neighborhood is $\Theta(n^k)$

Improving local search algorithms

- Initial solution can be randomly generated or can be a known solution found by some other method.
- When no improvement is possible restart the search from some other initial solution.
- Instead of a complete restart, can do a larger perturbation to the best solution found so far (iterated local search).
- Use population-based search algorithms.