COEN279 - Design and Analysis of Algorithms

Approximation Algorithm

Dr. Tunghwa Wang

Fall 2016

Announcement

- Final exam:
  - Like Mid-term exam, it is open book:
    - No electronic devices allowed.
  - Seat is assigned.
    - 25 students in class
  - Final grade:
    - As long as you finished all bonus and programming assignments correctly, you will definitely get B grade.
    - Exams are for finding out how much you have learned from this class.
  - Multi-part questions: Do not get blocked by any sub-question
    - Questions could be related: part 2 uses result from part 1.
    - Not knowing how to solve part 1 does not stop you from answering part 2 correctly: You may assume that part 1 is valid.

Announcement

- Final exam:
  - Strategy:
    - Make sure that you can do some questions correctly.
    - Questions are easy enough to be solved in 10-20 minutes.
  - 6 questions:
    - Greedy algorithm
    - Amortized analysis
    - Graph algorithm – BFS, DFS, Minimum Spanning Tree, and network flow
    - NP-complete complexity
- Last class 12/06:
  - Get your final papers back.
  - Summarize this class.
  - Feedback on this class.

NP-Complete Problems

- Finding optimal solution is NP-complete:
  - For application n is small: We may live with it.
    - In fact for non-NP problems, $O(2^n)$ algorithms may perform better in the ranges of small n.
    - However, we many frequently deal with very large n:
      - Network as graph with millions of nodes
      - Logic reasoning with large clause space
  - For application only special subset is targeted:
    - We may find lower complexity algorithm just to solve such subset.
  - For application we can live with near-optimal solutions:
    - We may find lower complexity algorithm just to solve such subset.
    - However, many such approximation versions are still NP-complete.

Approximation Algorithms

- We are working on optimization problem in which each potential problem has a cost and we hope to find a near-optimal solution.
  - Approximation ratio ρ(n):
    - C(n): the cost of solution generated by approximation algorithm
    - C*(n): the cost of the optimal solution
    - Maximization problem: $0 \le C(n) \le C^*(n)$
    - Minimization problem: $0 \le C^*(n) \le C(n)$
    - $1 \le \rho(n) = \max \{ C(n)/C^*(n), C^*(n)/C(n) \}$
  - ρ(n)-approximation algorithm:
    - Optimal solution may not be unique.
  - 1-approximation algorithm gives the best approximation.
  - The larger the ρ(n) is, the worse the approximation is.

Approximation Scheme

- It is an approximation algorithm with parameter $\varepsilon > 0$ to give $(1+\varepsilon)$-approximation.
  - Polynomial-time approximation scheme:
    - An approximation scheme such that for any fixed $\varepsilon > 0$, the scheme runs in time polynomial in the size n of its input instance.
  - Fully polynomial-time approximation scheme:
    - An approximation scheme such that for any fixed $\varepsilon > 0$, the scheme runs in time polynomial in both $1/\varepsilon$ and the size n of its input instance.

Vertex Cover

- VERTEX-COVER is NP-complete:
  - APPROX-VERTEX-COVER is a polynomial 2-approximation:
    - A vertex cover of an undirected graph G = { V, E } is a subset $V' \subseteq V$ such that if $(u, v) \in E$ then either $u \in V'$ or $v \in V'$ (or both).
    - The vertex cover problem is
      - Optimization problem of finding a vertex cover of minimum size in a graph.
      - Decision problem of whether a vertex cover of size k exists.
      - VERTEX-COVER = { <G, k>: G is a graph containing a vertex cover of size k }
  - Approximate vertex-cover algorithm:
    - We can find near-optimal solution in polynomial time: O(|V|+|E|)

```
     approximate_vertex_cover(G)
1    C = ∅
2    E' = G.E
3    while E' ≠ ∅
4         let (u, v) be an arbitrary edge of E'
5         C = C ∪ { u, v }
6         remove from E' every edge incident on either u or v
7    return C
```

Vertex Cover

- VERTEX-COVER is NP-complete:
  - APPROX-VERTEX-COVER s a polynomial 2-approximation:
    - Algorithm runs polynomial time.
    - C is a vertex cover of E.
    - $|C|/|C^*| \leq 2$:
      - Let A = { e: e chosen by line 4 of pseudocode }.
      - $|A| \leq |C^*|$ since no two edges of A can be covered by the same vertex of C*.
      - $|C| = 2|A|$ since both $u \in V$ and $v \in V$ must be true for all $(u, v) \in A$.
      - Thus, $|C| \leq 2|C^*|$.

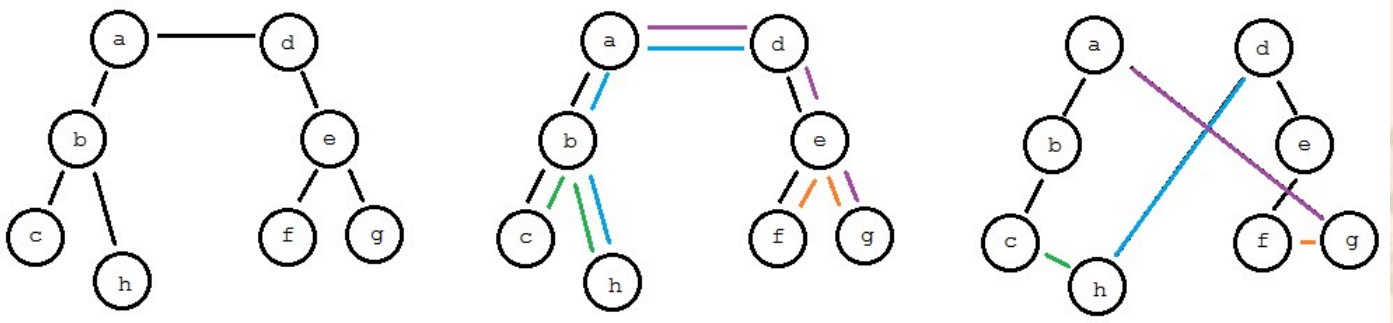
Traveling Salesman

- TSP is NP-complete:
  - Definition:
    - A salesman wishes to make a trip to visit n cities by visiting each city exactly once and come back to the starting city. Each trip from city u to city v has a nonnegative cost c(u, v). The salesman has a total budget limit of k and wish to incur minimum travel expenses.
    - The traveling-salesman problem is
      - Optimization problem of finding the minimum budget needed.
      - Decision problem of whether a such trip is possible with the budget limit.
      - TSP = { <G>: G is a complete graph, c: VxV -> N, k N, and G has a traveling-salesman tour with cost at most k }
  - Approximate traveling-salesman algorithm:
    - Cost function must satisfy triangular inequality condition.
    - We can find near-optimal solution in polynomial time: $O(|V|^2)$
    - Since G is complete, $O(|E|+|V|\lg|V|) = O(|V|^2)$

```
    approximate_traveling_salesman(G, c)
1   select a vertex r ∈ G.V as root.
2   compute a minimum spanning tree T for G from r using Prim's algorithm.
3   let H be a list of vertices ordered according to when visited in a preorder traversal.
4   return the Hamiltonian cycle H.
```

Traveling Salesman

- TSP is NP-complete:
  - APPROX- TSP s a polynomial 2-approximation:
    - Algorithm runs polynomial time.
    - H, as a Hamiltonian cycle of G, visits all vertices once.
    - c(|H|)/c(|H*|) ≤ 2:
      - Let T be the minimum spanning tree.
      - c(|T|) ≤ c(|H*|) since T is minimum.
      - For any Walk W, c(|W|) = 2c(|T|) since All edges in T are traversed twice e ∈ G.E.
      - c(|H|) ≤ c(|W|) = 2c(|T|) from triangular inequality.
      - Thus, c(|H|) ≤ 2c(|H*|).
  - Example:



Subset Sum

- SUBSET-SUM is NP-complete:
  - Definition:
    - S is a finite set of positive integers and a target integer t > 0. We want to find out whether there exists a subset $S' \subseteq S$ such that the sum of elements in S' is t.
    - The subset-sum problem is
      - Optimization problem of finding the maximal sum less than t.
      - Decision problem of whether a such subset exists giving the sum t.
      - SUBSET-SUM = { <S, t>: there exists a subset $S' \subseteq S$ such that $t = \Sigma_{x \in S'} x$ }
  - Approximate subset-sum algorithm:
    - It is fully polynomial-time approximation scheme.
    - Use a trim function to make sure that each list to work on is O(|S|) instead of $O(2^{|S|})$.

Subset Sum

- SUBSET-SUM is NP-complete:
  - Exact algorithm: $O(2^{|S|})$

```
     subset_sum(S, t)
1    n = |S|
2    L0 = { 0 }
3    for k = 0 to n
4         Lk = merge_lists(Lk-1, Lk-1+ xk)
5         remove from Lk every element that is greater than t
6    return the largest elemet of Ln
```

  - Approximate algorithm: $O(|S|)$

```
     approximate_subset_sum(S, t, ε)
1    n = |S|
2    L0 = { 0 }
3    for k = 0 to n
4         Lk = merge_lists(Lk-1, Lk-1+ xk)
5         Lk = trim_list(Lk, ε/2n)
6         remove from Lk every element that is greater than t
7    return the largest elemet of Ln
     trim_list(L, ε)
1    let m be the length of L = { y1, y2, ..., ym}
2    L' = { y1 }
3    last = y1
4    for k = 2 to m
5         if yk > last * (1 + ε)
6              append yk onto the end of L'
7              last = yk
8    return L'
```

Combinations

- Textbook also lists several approximate algorithms using other techniques learned from this class:
  - A greedy approximation algorithm: set-cover problem
  - A randomized approximation algorithm: MAX-3-CNF satisfiability problem
  - A linear programming approximation algorithm: weighted vertex-cover problem

Backup Slides