CS 362, Lecture 24

Jared Saia
University of New Mexico

Today's Outline

- Approximation algorithms for NP-Hard Problems

Vertex Cover

- A *vertex cover* of a graph is a set of vertices that touches every edge in the graph
- The decision version of *Vertex Cover* is: "Does there exist a vertex cover of size k in a graph G?".
- We've proven this problem is NP-Hard by an easy reduction from Independent Set
- The *optimization* version of *Vertex Cover* is: "What is the minimum size vertex cover of a graph G?"
- We can prove this problem is NP-Hard by a reduction from the decision version of Vertex Cover (left as an exercise).

Approximating Vertex Cover

- Even though the optimization version of Vertex Cover is NP-Hard, it's possible to *approximate* the answer efficiently
- In particular, in polynomial time, we can find a vertex cover which is no more than 2 times as large as the minimal vertex cover

Approximation Algorithm

- The approximation algorithm does the following until G has no more edges:
- It chooses an arbitrary edge (u, v) in G and includes both u and v in the cover
- It then removes from G all edges which are incident to either u or v

Approximation Algorithm

```
Approx-Vertex-Cover(G){
  C = {};
  E' = Edges of G;
  while(E' is not empty){
    let (u,v) be an arbitrary edge in E';
    add both u and v to C;
    remove from E' every edge incident to u or v;
  }
  return C;
}
```

Analysis

- If we implement the graph with adjacency lists, each edge need be touched at most once
- Hence the run time of the algorithm will be $O(|V| + |E|)$, which is polynomial time
- First, note that this algorithm does in fact return a vertex cover since it ensures that every edge in G is incident to some vertex in C
- Q: Is the vertex cover actually no more than twice the optimal size?

Analysis

- Let A be the set of edges which are chosen in the first line of the while loop
- Note that no two edges of A share an endpoint
- Thus, *any* vertex cover must contain at least one endpoint of each edge in A
- Thus if $C*$ is an optimal cover then we can say that $|C*| \geq |A|$
- Further, we know that $|C| = 2|A|$
- This implies that $|C| \leq 2|C*|$

Which means that the vertex cover found by the algorithm is no more than twice the size of an optimal vertex cover.

TSP

- An optimization version of the TSP problem is: “Given a weighted graph G, what is the shortest Hamiltonian Cycle of G?”
- This problem is NP-Hard by a reduction from Hamiltonian Cycle
- However, there is a 2-approximation algorithm for this problem if the edge weights obey the *triangle inequality*

Triangle Inequality

- In many practical problems, it’s reasonable to make the assumption that the weights, c, of the edges obey the *triangle inequality*
- The triangle inequality says that for all vertices $u, v, w \in V$:

$$c(u, w) \leq c(u, v) + c(v, w)$$

- In other words, the cheapest way to get from u to w is always to just take the edge (u, w)
- In the real world, this is usually a pretty natural assumption. For example it holds if the vertices are points in a plane and the cost of traveling between two vertices is just the euclidean distance between them.

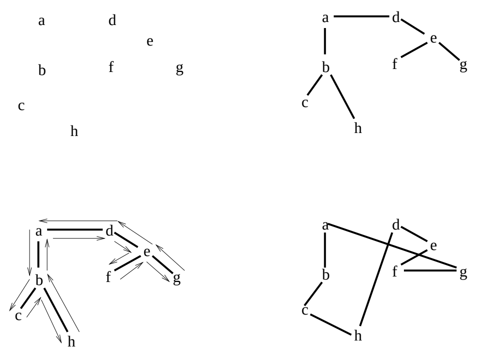
Approximation Algorithm

- Given a weighted graph G, the algorithm first computes a MST for G, T, and then arbitrarily selects a root node r of T.
- It then lets L be the list of the vertices visited in a depth first traversal of T starting at r.
- Finally, it returns the Hamiltonian Cycle, H, that visits the vertices in the order L.

Approximation Algorithm

```
Approx-TSP(G){
  T = MST(G);
  L = the list of vertices visited in a depth first traversal
      of T, starting at some arbitrary node in T;
  H = the Hamiltonian Cycle that visits the vertices in the
      order L;
  return H;
}
```

Example Run



The top left figure shows the graph G (edge weights are just the Euclidean distances between vertices); the top right figure shows the MST T. The bottom left figure shows the depth first walk on T, $W = (a, b, c, b, h, b, a, d, e, f, e, g, e, d, a)$; the bottom right figure shows the Hamiltonian cycle H obtained by deleting repeat visits from W, $H = (a, b, c, h, d, e, f, g)$.

Analysis

- The first step of the algorithm takes $O(|E| + |V| \log |V|)$ (if we use Prim's algorithm)
- The second step is $O(|V|)$
- The third step is $O(|V|)$.
- Hence the run time of the entire algorithm is polynomial

Analysis

An important fact about this algorithm is that: *the cost of the MST is less than the cost of the shortest Hamiltonian cycle.*

- To see this, let T be the MST and let $H*$ be the shortest Hamiltonian cycle.
- Note that if we remove one edge from $H*$, we have a spanning tree, T'
- Finally, note that $w(H*) \geq w(T') \geq w(T)$
- Hence $w(H*) \geq w(T)$

Analysis

- Now let W be a depth first walk of T which traverses each edge exactly twice (similar to what you did in the hw)
- In our example, $W = (a, b, c, b, h, b, a, d, e, f, e, g, e, d, a)$
- Note that $c(W) = 2c(T)$
- This implies that $c(W) \leq 2c(H*)$

Analysis

- Unfortunately, W is not a Hamiltonian cycle since it visits some vertices more than once
- However, we can delete a visit to any vertex and the cost will not increase *because of the triangle inequality*. (The path without an intermediate vertex can only be shorter)
- By repeatedly applying this operation, we can remove from W all but the first visit to each vertex, without increasing the cost of W.
- In our example, this will give us the ordering $H = (a, b, c, h, d, e, f, g)$

Analysis

- By the last slide, $c(H) \leq c(W)$.
- So $c(H) \leq c(W) = 2c(T) \leq 2c(H*)$
- Thus, $c(H) \leq 2c(H*)$
- In other words, the Hamiltonian cycle found by the algorithm has cost no more than twice the shortest Hamiltonian cycle.

Take Away

- Many real-world problems can be shown to not have an efficient solution unless $P = NP$ (these are the NP-Hard problems)
- However, if a problem is shown to be NP-Hard, all hope is not lost!
- In many cases, we can come up with an provably good approximation algorithm for the NP-Hard problem.

Final Review Session

- Final Review session will be *in class* on Thursday Dec. 9th
- Please come with questions

Final Info

- Final exam will be Tuesday Dec. 14th from 12:30-2:30 in our regular classroom.
- You can bring 2 pages of “cheat sheets” to use during the exam. You can also bring a calculator. Otherwise the exam is closed book and closed note.
- Note that the web page contains links to prior classes and their tests. *Many of my questions will be similar in flavor to these past tests!*

Final

- 5 to 6 questions
- There will be some time pressure, so make sure you can solve problems both quickly and correctly.
- I expect a class mean of between 60 :(and 70 :) points

Topics Covered

- Asymptotic Analysis and Recurrence Relations (Chapter 3 and 4 in text) : defns of big-O and friends, recursion trees, master method, annihilators and change of variables
- Dynamic Programming: general concepts, String Alignment, Matrix Multiplication, Longest Common Subsequence (Chapter 15)
- Greedy Algorithms: general concepts, activity selection, fractional knapsack, MST (Chapter 16)
- Amortized Analysis: Aggregate Method, Accounting Method, Potential Method, Dynamic Array (Chapter 17)
- Disjoint-Sets: Disjoint Set Operations, Representation as Forest, Union by Rank and Path Compression, Amortized Costs (Chapter 21)
- Minimum Spanning Trees: Definition, Kruskal’s Algorithm, Prim’s Algorithm, Safe Edge Theorem and Corollary

- Graph Algorithms: Graph Representations, BFS, DFS, Single-Source Shortest Path, All-Pairs Shortest Paths; Dijkstra’s, Bellman-Ford, Floyd-Warshall (Chapters 22 23,24,25)
- NP-Hard Problems: Definitions of P, NP, co-NP, NP-Hard, and NP-Complete; General concepts; Reductions (i.e. how to show that a problem is NP-Hard); Classic NP-Hard problems: Circuit Satisfiability, SAT, 3-SAT, Coloring, Clique, Vertex Cover, Independent Set, Hamiltonian Cycle, TSP. (Chapter 34)
- Approximation Algorithms: Vertex Cover, TSP, etc.

In general should know the resource bounds for all algorithms covered.

Example Problem - Short Answer

Collection of true/false questions, matching and short answer questions. Some examples:

- T/F questions covering all topics
- Multiple Choice e.g. I give you some “real world” problems and ask you which algorithm we’ve studied in class that you would use to solve each of them; I give you some problems and ask you how fast they can be solved, etc.

Example Problem - Review

- Possibility 1: Asymptotic Analysis / Recurrence Relations
- Possibility 2: Dynamic Programming (new: *Floyd-Warshall*, Dijkstra’s, and Bellman-Ford)
- Possibility 3: Greedy Algorithms (new: Kruskal’s and Prim’s)
- Possibility 4: Amortized Analysis

Example Problem - Graph Theory

- Possibility 1: BFS or DFS algorithms (and how to use them)
- Possibility 2: Single Source Shortest Paths (Dijkstra’s and Bellman-Ford)
- Possibility 3: All Pairs Shortest Paths (Floyd-Warshall)

Example Problem - NP-Hardness

- Possibility 1: Something like Challenge problem 1 from last lecture
- Possibility 2: I give you a problem and ask you to prove it’s NP-Hard by a reduction from another NP-Hard Problem.
- Possibility 3: I give you an NP-Hard Problem and ask you give an approximation algorithm for it (the problem would be a variant of something already seen in class)