Advanced Algorithms

Lecture by Shay Mozes
Typeset by Steven Karas

2016-11-27
Last edited 18:19:03 2016-12-28

1 TSP Approximation

MST approximation The pre-order DFS tour of the MST of an Euclidean graph gives an approximation that is within x2 of the optimum.

More generally, we use the DFS of the MST to find an approximate tour of a subgraph.

1.1 Euler Tours

An Euler tour is a tour through a graph that passes through every edge exactly once. Well known that an undirected graph contains an Euler tour iff every vertex has an even degree. It's trivial to construct Euler tours.

Note that if there isn't a Euler tour, the number of odd-degree vertices is even![1]

Maximum Matching A matching in a graph $G = (V, E)$ is a subset of edges M from E such that each vertex in V is incident to at most one edge in M. A maximum matching in G is a matching of maximum cardinality. A minimum weight matching in a weighted graph is a maximum matching of minimum total weight. Finding such a matching can be done in poly-time.

We treat this as a black box

1.2 Christofides Algorithm

1. Find a MST T
2. Find a minimum weight matching M in the subgraph induced by odd-degree vertices in T
3. Find an Euler tour of $T \cup M$
4. Make shortcuts

A tour always exists

[1]Proven on whiteboard

Approximation factor Christofides algorithm finds a tour of length at most 3/2 optimal. To prove this, we will show that the weight of the $MST \leq OPT$, that the weight of the matching $\leq OPT/2$, and that the shortcuts do not increase the cost.

If we take the matching, and draw shortcuts between them. Because these shortcuts cannot worsen an optimal tour, and they make up half the cycle of M. Note that even if the matching is itself a shortcut across the optimal path, they are no worse than the optimal path (by the triangle inequality). Therefore, the weight of this matching is no heavier than $1/2\, OPT$.

1.3 Non-metric TSP

We will show that without the triangle inequality (as a metric on the graph), that if we can provide a c-approximation for TSP, then $P = NP$.

Proof Reduction from Hamiltonian cycle to TSP. Assume that TSP is c-approximable. Given $G = (V, E)$ and the question "is there a HC in G?", we construct a TSP instance G', such that if there is a HC in G then there is a TS tour of cost at most n in G', and if there is no HC in G, then the minimum TS tour has cost greater than cn.

Construct $G' = (V', E')$ as a clique, with the newly introduced edges having weight cn, and the existing edges set to 1.

Note that building G' takes poly-time, but that writing the edges with weight cn takes $O(\log n)$.

2 Bin Packing

Given a sequence of items $a_1, ..., a_n$ such that $\forall a : a_i \in (0, 1)$. The goal of bin packing is to place the items into bins of size 1, using as few bins as possible.

Example Given items 1/2, 1/3, 2/5, 1/6, 1/5, 2/5; An optimal packing is $(1/2, 1/3, 1/6)$ and $(2/5, 2/5, 1/5)$.

2.1 Next-fit Algorithm

Track an active bin, and open a new bin if the currently active bin cannot contain the next item.

Approximation This algorithm is approximate to within 2OPT.

Proof Note that $OPT \geq \sum_i a_i$. The sum of the items in two consecutive bins is greater than 1 (otherwise we would have put them together). Assume that $h \geq 2OPT$. Thus, the sum of items in the first $2OPT$ bins is greater than OPT. This contradicts that $OPT \geq \sum_i a_i$.

Analysis Given a list of $4n$ items $\{\frac{1}{2}, \frac{1}{2n}, \frac{1}{2}, \frac{1}{2n}, ...\}$, Next-fit will use $2n$ bins. An optimal packing in $n+1$ bins is possible: n bins with two items, and one last bin with all the small items.

$$\lim_{n\to\infty} \frac{2n}{(n+1)} \to 2$$

3 Knapsack

Given a knapsack with capacity W. Each item has a weight w_i and a value v_i.

3.1 Greedy algorithm

Sort the items by weighted value, and add them to the knapsack in order.

But this algorithm is not optimal, and not even bound. However, if the items are divisible, it is optimal.

Proof To get some ratio c, consider the following instance:

There are two items: $i_1 = (b = 2, w = 1)$ and $i_2 = (b = 2c, w = 2c)$.

3.2 2-Approx

Take the maximum of Greedy, and the largest value item that fits by itself.

Approximation Proof Recall that the greedy algorithm is optimal for divisible items. Thus: $OPT < Greedy +$ value of first item not packed by Greedy $<$ $Greedy +$ largest value B

We assume that all items have weight at most W, and that the items are sorted by decreasing value to weight ratio. Let B be the largest value, and G be the value computed by the greedy algorithm.

$$ALG = \max(B, G) \geq \frac{B+G}{2}$$

$$G = \sum_{i=1}^{j-1} b_i$$

$$B \geq b_j$$

$$G + B \geq \sum_{i=1}^{j} b_i > OPT$$

Which all implies that $ALG > OPT/2$

3.3 Dynamic Programming

Let $A[i,p] =$ minimum weight of a subset of items $1, ..., i$ whose total value is exactly p. Note that $A[i,p] = \infty$ if there is no such subset. Note that $i = 1, ..., n$, $p = 1, ..., nB$

$A[1,p]$ is trivial to compute.

$$A[i+1,p] = \min(A[i,p],\ w_{i+1} + A[i, p - b_{i+1}])$$

$OPT =$ maximum p for which $A[i,p] \leq W$

Complexity $O(n^2B)$

Representation matters Knapsack: the input is pairs of weights and values as binary integers. Input size of $\Theta(n \log B)$

Unary-Knapsack: the input is pairs of weights and values in unary. Input size of $\Theta(nB)$

Note that $n^2B = O((nB)^2)$, meaning that dynamic programming is polynomial for Unary-Knapsack, meaning that Unary-Knapsack is in P.

Let $B = 2^n$. Thus, $O((nB)^2) = n^2$, yet n^2B is $\Omega(2^n)$, so the algorithm is not polynomial for Knapsack. This is not suprising, considering that Knapsack is NP-Complete.

Representation for TSP Let TSP's input size $x = \Theta(n^2 \log B)$, and Unary-TSP's input size $y = \Theta(n^2B)$. We can show that Unary-TSP is still NP-Complete, because a solution for Unary-TSP can solve for a Hamiltonian Cycle.

Weakly NP-Hard Problems that become easier (in P) when the input is unary are referred to as being weakly NP-Hard. Problems that don't are called strongly NP-Hard. Knapsack is weakly NP-Hard, and TSP is strongly NP-Hard.

4 Polynomial Time Approximation Schemes

A PTAS is an algorithm that takes an additional parameter ε. For any fixed ε, this runs in poly-time (in relation to n), and produces a solution whose value is at least $(1 \pm \varepsilon)OPT$ for optimization problems.

The dependency of running time to ε is arbitrary (e.g. $n^{1/\varepsilon}$ is ok because ε is fixed). This gives a clear trade-off between running time and quality of approximation.

5 PCP Theorem

We can trade off running time for more accuracy.