Script of Lecture 1, Approximation Algorithms
Summer term 2017

Saarland Informatics Campus

Tobias Mömke, Hang Zhou
http://www-cc.cs.uni-saarland.de/course/61/

Written by Tobias Mömke

1 Why Approximation Algorithms

An approximation algorithm is an algorithm that solves optimization problems not optimally but with *guaranteed* quality. Approximation algorithms are particularly interesting for NP-hard optimization problems since we cannot solve them efficiently (in polynomial time) unless $\mathsf{P} = \mathsf{NP}$. At the same time, we can efficiently check whether a given solution is feasible.

NP-hard optimization problems frequently appear in practical applications such as

- Scheduling in manufacturing
- Finding good layouts in VLSI integration
- Design of robust networks

2 Approximation

Given an optimization problem P with set of feasible solutions S (depending on the instances of P), we want to find a solution $s \in S$ s. t. the weight of s is optimal.

There is a difference whether we want to minimize or maximize.

An algorithm A is an α-approximation algorithm if for every instance of P,

- minimization:
  $w(s) \leq \alpha \cdot w(s')$
- maximization:
  $w(s) \geq \alpha \cdot w(s')$

for all $s' \in S$, where w specifies the weight of the solution and s is the solution computed by A. Equivalent definition: we require that s' is an optimal solution.

Note: Sometimes in the literature, for maximization problems α is replaced by $1/\alpha$ in order to have approximation ratios of at least one for both minimization and maximization problems. Rule of thumb: the closer the ratio is to one, the better. This is true for both minimization and maximization in both definitions.

3 Example: Traveling Salesman Problem (TSP)

Input: Complete graph $G = (V, E, w)$

Feasible Solutions: Each cycle such that each $v \in V$ is visited *exactly* once

Goal: Minimize the overall weight (i. e., for a solution/cycle C, $\sum_{e \in E(C)} w(e)$)

TSP is NP-hard: it can be seen as a generalization of the decision problem Hamiltonian Cycles (HC). (Difference between HC and TSP: in HC we are given an *unweighted* graph that in general is not complete. The goal is to decide whether the graph has a cycle that visits each vertex exactly once.)

4 General TSP is very hard

Theorem 1. *General TSP is inapproximable.*

Proof. We take an HC instance $G = (V, E)$ and construct a TSP instance $G' = (V, E', w')$. Since G' is a complete graph, $E \subseteq E'$. If $e \in E$, we set $w(e) := 1$. If $e \notin E$, we set $w'(e) := 2^n$, where $n = |V|$.

If G has an HC, there is a TSP solution in G' of weight n since each edge of the cycle has weight one. Otherwise, if G does not have an HC, every solution contains at least one edge of weight 2^n and therefore every solution is larger than 2^n.

If we find a TSP solution in G' of weight at most 2^n, we can conclude that G has an HC (since otherwise every solution is larger than 2^n).

If we would have a $2^n/n$-approximation algorithm for TSP, we would be able to decide whether G has an HC: If G has an HC, the algorithm would be able to compute a solution of weight at most $2^n/n \cdot n = 2^n$.

We conclude that it is NP-hard to find a $2^n/n$ approximation. (We always implicitly assume that approximation algorithms run in polynomial time.)

The value 2^n is chosen arbitrarily. Choosing larger values, however, may conflict with the computation model used since we have to encode the weights in the input, and we measure the running time with respect to the input length. □

5 Metric

A function $c\colon V \times V \to \mathbb{R}$ is a metric if for each triple $u, v, w \in V$,

a) $c(u, v) \geq 0$ (non-negativity)

b) $c(u, v) = 0$ iff $u = v$ (identity of indiscernibles)

c) $c(u, v) = c(v, u)$ (symmetry)

d) $c(u, w) \leq c(u, v) + c(v, w)$ (triangle inequality)

For graphs: usually we have a positive weight function. There is no essential difference between writing $w(e)$ and $c(u, v)$ for an edge $e = \{u, v\}$. The set notation implies symmetry. Therefore a) to c) are usually satisfied for undirected weighted graphs.

The important property to check is the triangle inequality. Figuratively, the triangle inequality states that in a triangle u, v, w the direct connection from u to w is not more expensive than the detour via v.

A metric graph is a complete graph with a weight function that is a metric (i. e., that satisfies the triangle inequality).

metric TSP: TSP in metric graphs.

Exercise (not graded): metric TSP is equivalent to general TSP with the relaxation that vertices may be visited multiple times.

6 Tree Doubling

Algorithm TreeDoubling
Input: a metric graph $G = (V, E, w)$.

a) Compute a minimum spanning tree (MST) T of G.

b) Double all edges in T, obtain $2T$.

c) run a depth first search (DFS) and "shortcut" solution to obtain a Hamiltonian cycle T'.

Output: T'

We have to specify the meaning of "shortcut." We choose a root of T and traverse T in DFS manner. During the traversal, we add the visited edges, where we use the second copy of an edge when visiting it a second time. Whenever we notice that there is a path P of length at least 2 such that all internal vertices are visited at least twice, we replace P by a direct edge between the ends. We note that newly introduced edges may be removed again in later steps.

The details regarding shortcutting are part of the exercises.

Theorem 2. TREEDOUBLING *is a 2-approximation algorithm for metric TSP.*

Proof. We show that the algorithm runs in polynomial time. Computing an MST can be done with Kruskal's algorithm in time $O(|E| \log |E|)$. The doubling takes $O(|V|)$ (note that T has $|V| - 1$ edges). The shortcutting is essentially a DFS search which also can be done in time $O(|V|)$.

Next we show that the algorithm is correct, i. e., it computes a feasible solution.

(a) T' is connected. Clearly T is connected since it is a tree in G. In the shortcutting, we only replace a path by an edge if all internal vertices have been visited twice. That is, when the DFS reaches a vertex v, all predecessors of v in the search are connected to v, also after the shortcutting. Since the DFS visits all vertices from V, in the end we still have a connected graph.

(b) T' is a cycle. Equivalently: each vertex has degree 2 within T'. The idea is that after the DFS has visited a vertex v, whenever it visits v again, the new visit results in a path that can be shortcut. The details are left as an exercise.

Finally, we analyze the approximation ratio. The weight of an MST is a lower bound on the weight of an optimal TSP solution: each TSP solution can be transformed into a tree by removing one edge.

Therefore the weight of $2T$ is at most twice the weight of an optimal TSP solution.

Due to the metric, shortcutting $2T$ to an HC can be done without increasing the weight: each edge e that replaces a path P has at most the weight of P. □

7 General View

An Eulerian tour in a graph is a circuit (i. e., cycle with vertex repetitions) that visits each edge exactly once. A graph is Eulerian if it has an Eulerian tour.

- We computed a tree T to obtain connectivity
- We modified T to obtain an Eulerian graph $2T$
- We computed an Eulerian tour in $2T$
- We transformed the tour into an HC

We will use the following results from graph theory (not part of the course):

- A graph G is Eulerian iff G is connected and even. (A graph is even if each vertex has even degree).
- We can compute an Eulerian tour in an Eulerian graph in polynomial time.

8 Matchings

Given a graph $G = (V, E)$, a set of edges M is a *matching* if every vertex $v \in V$ is incident to at most one edge $e \in M$.

M is a *perfect matching* of G if every $v \in V$ is incident to *exactly* one edge in M.

For a set $S \subseteq V$ of vertices, a perfect matching M of S in G is a matching such that each vertex in S is incident to exactly one edge of M and no vertex from $V \setminus S$ is incident to edges of M.

A minimum cost perfect matching can be computed in polynomial time. (We don't proof that in this course and use the result as a black box.)

If we want to obtain an Eulerian graph, we want to change the parity of the odd degree vertices. One way to achieve this is to add edges to the graph such that the degree of each odd degree vertex is increased by one.

An important result from graph theory is that in every graph, the number of odd degree vertices is even. (The reason is that each edge contributes an even number to the total sum of degrees in the graph). Since we consider complete graphs, we can add a perfect matching of the odd degree vertices of T and this way create an Eulerian graph.

9 Christofides' Algorithm

Input: A metric graph $G = (V, E, w)$

a) Compute an MST T

b) Compute a minimum cost perfect matching M on the odd degree vertices of T.

c) Compute an Eulerian tour T' in $T + M$ (the multi-graph obtained by adding the edges of M and T).

d) Shortcut T' to an HC T''

Output: T''

Theorem 3. *Christofides' algorithm is a* 1.5*-approximation algorithm for metric TSP.*