

Improving the ratio to $3/2$

Better-Approximate-TSP(G, c)

1. Compute a minimum spanning tree T of G .
2. Compute a minimum cost perfect matching M on the set of odd-degree vertices of T . Add M to T to obtain a Eulerian graph G' .
3. Compute an Euler tour of G' starting at an arbitrary vertex a .
4. Compute the TSP by starting at vertex a , following the Euler tour, skipping vertices that were already visited.

Proof

Lemma 1. *Let $V' \subseteq V$, such that $|V'|$ is even, and let M be a minimum cost perfect matching on V' . Then $\text{cost}(M) \leq OPT/2$.*

Proof. Let τ be an optimal tour of G . Let τ' be the tour on V' obtained by short-cutting τ .

$$\text{cost}(\tau') \leq \text{cost}(\tau).$$

τ' is the union of two perfect matchings on V' , each consisting of alternate edges of τ' .

Thus the cheaper of these matchings has cost $\leq \text{cost}(\tau')/2 \leq OPT/2$.

Hence the optimal matching has cost at most $OPT/2$. \square

Theorem 1. *Better-Approximate-TSP is a $3/2$ -approximation algorithm.*

Proof. The cost of the Euler tour is $\leq \text{cost}(T) + \text{cost}(M) \leq OPT + \frac{1}{2}OPT = \frac{3}{2}OPT$.

The cost of the tour is at most the cost of the Euler tour. \square

Limits on Approximation

Theorem 2. *If $P \neq NP$ then there is no polynomial time approximation algorithm for the general TSP problem for any $\rho \geq 1$.*

Proof.

Assume that we have such an approximation algorithm, we'll use it to solve the Hamiltonian problem.

Given a graph $G = (V, E)$, let G' be a complete graph with cost function

$$c(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ \rho|V| + 1 & \text{otherwise} \end{cases}$$

G has an Hamiltonian cycle, then G' has a TSP of cost $|V|$.

Any TSP solution in G' that is not an Hamiltonian cycle in G has cost at least

$$\rho|V| + 1 + |V| - 1 > \rho|V|.$$

Assume that we run an approximation algorithm AP with ratio bound ρ on G' :

If G has an Hamiltonian cycle, AP will return that path.

If G does not have an Hamiltonian cycle, AP will return a TSP with cost more than $\rho|V|$.

Thus, AP solves the Hamiltonian path problem in G . \square

Fully Polynomial-Time Approximation Scheme

A minimization (maximization) problem has a **fully polynomial-time approximation** scheme if for any $\epsilon > 0$ there is an $1 + \epsilon$ ($1 - \epsilon$) approximation algorithm for the problem that runs in time which is polynomial in both the problem size n and $1/\epsilon$.

Problems which have pseudo-polynomial time algorithms can be typically converted to yield FPTASs.

Knapsack

Given a set $S = \{a_1, a_2, \dots, a_n\}$ of n objects, with specified sizes and profits (all values are positive integers), and a knapsack capacity B , a positive integer. The problem is to find a subset of objects whose total size is bounded by B and the total profit is *maximized*. (Without loss of generality, assume that every object has size at most B .)

Admits a pseudo-polynomial time algorithm based on dynamic programming.

Let P be the profit of the most profitable object, i.e., $P = \max_{a \in S} \text{profit}(a)$. (Then nP is a trivial upper bound for any solution.)

For each $i \in \{1, \dots, n\}$ and $p \in \{1, \dots, nP\}$, let S_{ip} denote a subset of $\{a_1, \dots, a_i\}$ whose total profit is exactly p and whose total size is minimized.

Let $A(i, p)$ denote the sum of the sizes of the elements in the set S_{ip} ($A(i, p) = \infty$ if no such set exists).

The recursive formulation is:

$$A(i + 1, p) = \min\{A(i, p), \text{size}(a_{i+1}) + A(i, p - \text{profit}(a_{i+1}))\} \text{ if } \text{profit}(a_{i+1}) < p.$$
$$= A(i, p) \text{ otherwise.}$$

and $A(1, p)$ can be trivially computed for every $p \in \{1, \dots, nP\}$.

All values $A(i, p)$ can be computed in $O(n^2P)$ time.

The maximum profit achievable by objects of total size bounded by B is $\max\{p \mid A(n, p) \leq B\}$.

FPTAS for Knapsack

Idea: Ignore a certain number of least significant bits of profits depending on the error parameter ϵ .

FPTAS:

1. Given $\epsilon > 0$, let $K = \frac{\epsilon P}{n}$.
2. For each object a_i define $profit'(a_i) = \lfloor profit(a_i)/K \rfloor$.
3. Run the dynamic programming algorithm with the new profits and output the most profitable set.

Lemma 2. Let A denote the set output by the algorithm. Then

$$\textit{profit}(A) \geq (1 - \epsilon)OPT$$

Proof. Let O denote the optimal set.

For any object a , because of the rounding down,

$$\textit{profit}(a) - K \times \textit{profit}'(a) \leq K.$$

Hence, $\textit{profit}(O) - K \times \textit{profit}'(O) \leq nK$.

Dynamic programming must return a set at least as good as O under the new profits. Hence

$$\textit{profit}(A) \geq K \times \textit{profit}'(O) \geq \textit{profit}(O) - nK = OPT - \epsilon P \geq (1 - \epsilon)OPT$$

since $OPT \geq P$. \square

Theorem 3. *The above algorithm is an FPTAS for Knapsack.*

Proof. The solution is within $(1 - \epsilon)OPT$.

The running time of the algorithm is

$$O(n^2 \lfloor \frac{P}{K} \rfloor) = O(n^2 \lfloor \frac{n}{\epsilon} \rfloor)$$

which is polynomial in n and $1/\epsilon$. \square

Linear Programming

Let A be an $k \times n$ matrix. $\bar{x} \in \mathbb{R}^n, \bar{c} \in \mathbb{R}^n, \bar{b} \in \mathbb{R}^k$.

Maximize (Minimize)

$$G(\bar{x}) = \bar{c}^T \bar{x}$$

Subject to

$$A\bar{x} \leq \bar{b}$$

A setting of the variables x_1, x_2, \dots, x_n that satisfies all the constraints is called a **feasible** solution; a setting where even one of the constraints is not satisfied is called **infeasible**.

A solution \bar{x} has objective value $\bar{c}^T \bar{x}$. A feasible solution whose objective value is maximum over all solutions is an **optimal solution** to the LP.

Each constraint

$$\sum_{j=1}^n a_{i,j} x_j \leq b_i$$

defines a half space of R^n .

Each

$$\sum_{j=1}^n a_{i,j} x_j = b_i$$

is a hyperplane.

A finite intersection of hyperplanes defines a polyhedron.

We are looking for a point in the polyhedron defined by

$$A\bar{x} \leq \bar{b}$$

that maximizes (minimizes)

$$G(\bar{x}) = \bar{c}^T \bar{x}$$

There is a **polynomial-time** algorithm for solving an LP.

However if we restrict the solution vector to be integers (even 0 or 1) then solving the problem - called **Integer LP** - is NP-complete.