Fundamental Algorithms

The Last Chapter: Efficiency Beyond Efficiency

Jan Křetínský

Winter 2017/18
Plan

- Hard Problems
- Approximation of NP-Complete Problems
NP-hard Problems

- not believed to be “efficiently” solvable, i.e., in polynomial time
- **NP-complete**: many combinatorial/graph problems, satisfiability of a propositional-logic formula (SAT)
- even harder: many problems in AI, verification, …

Today: What to do with NP-complete problems?
NP-hard Problems

- not believed to be “efficiently” solvable, i.e., in polynomial time
- **NP-complete**: many combinatorial/graph problems, satisfiability of a propositional-logic formula (SAT)
- even harder: many problems in AI, verification, . . .

**Today:** What to do with **NP-complete** problems?
- more computational power?
NP-hard Problems

• not believed to be “efficiently” solvable, i.e., in polynomial time
• **NP-complete**: many combinatorial/graph problems, satisfiability of a propositional-logic formula (SAT)
• even harder: many problems in AI, verification, …

**Today: What to do with NP-complete problems?**

• more computational power?
• encode into SAT
NP-hard Problems

- not believed to be “efficiently” solvable, i.e., in polynomial time
- **NP-complete**: many combinatorial/graph problems, satisfiability of a propositional-logic formula (SAT)
- even harder: many problems in AI, verification, ...

**Today: What to do with NP-complete problems?**

- more computational power?
- encode into SAT
- approximation algorithms
Travelling Salesman Problem

Definition (TSP)

Given a complete, weighted, undirected graph $G = (V, E)$ with non-negative weights $c: V \rightarrow \mathbb{N}$, find a cycle that visits exactly all nodes and does so with minimal length.
Travelling Salesman Problem

Definition (TSP)

Given a complete, weighted, undirected graph \( G = (V, E) \) with non-negative weights \( c : V \rightarrow \mathbb{N} \), find a cycle that visits exactly all nodes and does so with minimal length.

Properties

- We can assume triangle inequality:
  \[ \forall u, v, w \in V. c(u, v) \leq c(u, w) + c(w, v) \]
- NP-complete
- We show a 2-approximation
- There is a 1.5-approximation
- There is no \( \frac{123}{122} \)-approximation (since 2015)
Travelling Salesman Problem

Definition (TSP)

Given a **complete**, weighted, undirected graph $G = (V, E)$ with non-negative weights $c: V \rightarrow \mathbb{N}$, find a cycle that visits exactly all nodes and does so with **minimal length**.

Properties

- We can assume **triangle inequality**:

  $$\forall u, v, w \in V. c(u, v) \leq c(u, w) + c(w, v)$$

- NP-complete
- We show a 2-approximation
- There is a 1.5-approximation
Travelling Salesman Problem

Definition (TSP)
Given a complete, weighted, undirected graph $G = (V, E)$ with non-negative weights $c : V \to \mathbb{N}$, find a cycle that visits exactly all nodes and does so with minimal length.

Properties
- We can assume triangle inequality:
  \[ \forall u, v, w \in V. c(u, v) \leq c(u, w) + c(w, v) \]
- NP-complete
- We show a 2-approximation
- There is a 1.5-approximation
- There is no $\frac{123}{122}$-approximation (since 2015)
2-Approximation Algorithm for TSP

Algorithm

1. $T :=$ a minimum spanning tree
2. $\text{cycle} :=$ traverse along depth-first search of $T$, jumping over visited nodes
2-Approximation Algorithm for TSP

Algorithm

1. T := a minimum spanning tree
2. cycle := traverse along depth-first search of T, jumping over visited nodes

Algorithm is

- polynomial
- 2-approximation
  - \( c(T) \leq \text{minimal cycle} \)
  - traversal costs \( 2 \cdot c(T) \) since jumping over costs at most the sum of traversed edges
Knapsack

Definition (TSP)

Given weight $W$ of knapsack and weights and values of $n$ items: $w_1, \ldots, w_m, v_1, \ldots, v_n$, pick $I \subseteq \{1, \ldots\}$ such that $\sum_{i \in I} w_i \leq W$ and $\sum_{i \in I} v_i$ is maximal (under the previous constraint).
Knapsack

**Definition (TSP)**

Given weight $W$ of knapsack and **weights** and **values** of $n$ items: $w_1, \ldots, w_m, v_1, \ldots, v_n$, pick $I \subseteq \{1, \ldots\}$ such that $\sum_{i \in I} w_i \leq W$ and $\sum_{i \in I} v_i$ is maximal (under the previous constraint).

**Greedy Algorithm**

- take items in the order $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \cdots \geq \frac{v_n}{w_n}$
Knapsack

Definition (TSP)

Given weight $W$ of knapsack and weights and values of $n$ items: $w_1, \ldots, w_m, v_1, \ldots, v_n$, pick $I \subseteq \{1, \ldots\}$ such that $\sum_{i \in I} w_i \leq W$ and $\sum_{i \in I} v_i$ is maximal (under the previous constraint).

Greedy Algorithm

- take items in the order $v_1 / w_1 \geq v_2 / w_2 \cdots \geq v_n / w_n$

Properties

- optimal for “fractional” knapsack problem
- for $v_1 = 1.001$, $w_1 = 1$, $v_2 = W$, $w_2 = W$ no better than a $W$-approximation.
2-Approximation of Knapsack

**Modified Greedy Algorithm (ModGreedy):**

- $S_1 :=$ solution by Greedy
- $S_2 :=$ item with the largest value
- Return whichever of $S_1, S_2$ that has more value

**Lemma**

*ModGreedy is a 2-approximation.*
2-Approximation of Knapsack

Modified Greedy Algorithm (ModGreedy):

- $S_1 := \text{solution by Greedy}$
- $S_2 := \text{item with the largest value}$
- Return whichever of $S_1, S_2$ that has more value

Lemma

*ModGreedy is a 2-approximation.*

Proof.

- If Greedy takes items 1, 2, \ldots, $k - 1$, then
  \[ \sum_{i=1}^{k} v_i \geq \text{OPT}_{\text{frac}} \geq \text{OPT} \]: $k$th item might not be taken in full + the optimal integral solution is not better than the optimal fractional solution
- \[(v_1 + \cdots + v_{k-1}) + v_k \geq \text{OPT}\]
- one of the two is $\geq \text{OPT}/2$
- $v(S_1) = \sum_{i=1}^{k-1} v_i$, and $v(S_2) = v_{\text{max}} \geq v_k$
PTAS for Knapsack

- Polynomial-time approximation scheme (PTAS): any approximation ratio possible
- Idea: brute-force a part of the solution and then use Greedy Algorithm to finish up the rest

**Algorithm**, $k$ fixed constant

- for all possible subsets of objects that have up to $k$ objects:
  - use the greedy algorithm to fill up the rest of the knapsack
- return the most profitable subset
PTAS for Knapsack

- Polynomial-time approximation scheme (PTAS): any approximation ratio possible
- Idea: brute-force a part of the solution and then use Greedy Algorithm to finish up the rest

Algorithm, \( k \) fixed constant

- for all possible subsets of objects that have up to \( k \) objects:
  - use the greedy algorithm to fill up the rest of the knapsack
- return the most profitable subset

Properties

- runtime \( \mathcal{O}(kn^k) \) subsets, filling up in \( \mathcal{O}(n) \)
- thus total running time \( \mathcal{O}(kn^{k+1}) \)
- \((1 + \frac{1}{k})\)–approximation