Reading: 11.0-11.3

Last time:

- greedy by value
- MST correctness.
- matroids

Today:

- approximation
- metric TSP
- knapsack

Approximation Algorithms

"instead of computing an optimal solution is $\mathcal{NP}$-complete, try to compute an approximately optimal solution instead"

Def: $\mathcal{A}$ is an β-approximation the value of its solutions is at most $\beta\mathrm{O}PT$ (minimization problems)

(at most $\mathrm{O}PT/\beta$ for maximization problems)

Question: how well can we approximate NP-complete problems?

$1+\epsilon$	const	log	linear	inapprox
Knapsack	METRIC-TSP			TSP

Metric TSP

Def: distances are a metric if

- symmetry: $d(u, v) = d(v, u)$
- triangle inequality: $d(u, v) \leq d(u, w) + d(w, v)$

Def: Metric TSP = TSP when edge costs are a metric.

Lemma: MST is smaller than TSP tour.

Proof:

- take any tour
- remove one edge

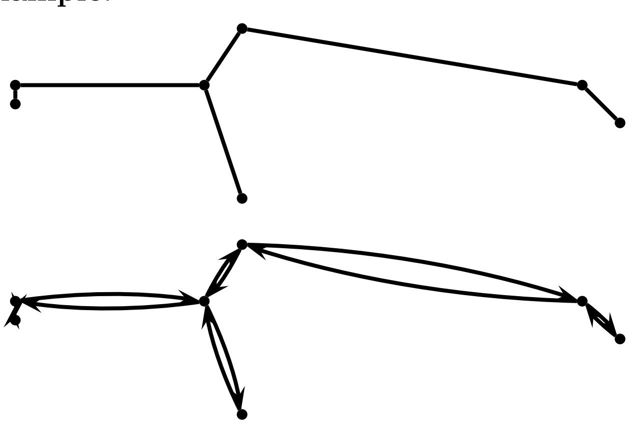
$\Rightarrow$ get a tree (degerate = a line)

$\Rightarrow$ cost of tour > cost of MST.

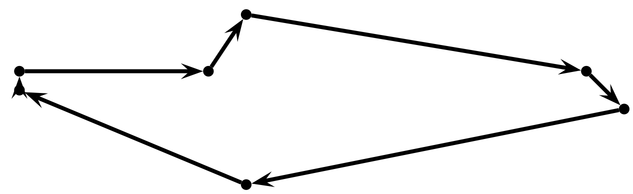
Algorithm: METRIC-TSP via MST

1. find MST.
2. double it $\Rightarrow$ cycle
   (with repeated vertices)
3. remove repeated vertices (short-cut) $\Rightarrow$ tour.

Example:



Cycle: ?



Challenge:

- $\mathcal{NP}$-hardness $\Rightarrow$ don't understand optimal soln's.
- how can we approximate something we don't understand?

Approach

1. Bound OPT. E.g., OPT $\geq$ MST
2. Design alg to approximate bound. E.g., $\mathcal{A} \leq$ 2MST.

Question: can we approximate (non-metric) TSP?

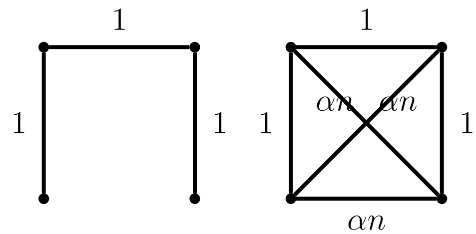
Lemma: Cannot approximate TSP to any factor unless $\mathcal{P} = \mathcal{NP}$.

Proof: reduce from Hamiltonian Cycle to α-approximate-TSP

- convert HC problem $G' = (V', E')$ to TSP problem $G, c(\cdot)$
- $G \leftarrow$ complete graph on V'.
- set $c(e) = \begin{cases} 1 & \text{if } e \in E' \\ \alpha n & \text{otherwise} \end{cases}$
- HC in $G' \Rightarrow$ TSP of cost n.
- no HC in $G' \Rightarrow$ TSP of cost $> \alpha n$.
- α-approxiate TSP distinguishes these two cases.

QED

Example:



Knapsack

input:

- n objects
- sizes s_i (non-negative real number)
- values v_i
- capacity C.

output: subset S that

- fits: $\sum_{i \in S} s_i \leq C$
- maximizes values: $\sum_{i \in S} v_i$.

Note: Knapsack is $\mathcal{NP}$-complete

Goal: approximation algorithm for knapsack

Step 0: try things that don't work.

Idea: Greedy by value/size

Example: $\mathbf{v} = (2, C)$, $\mathbf{s} = (1, C)$

Greedy $\Rightarrow$ 2; OPT $\Rightarrow$ C.

Step 1: find upper bound.

Fact: optimal fractional knapsack (FOPT) $\geq$ optimal integral knapsack (OPT)

Step 2: find algorithm to approximate upper bound.

Note: the difference between FOPT and GREEDY is that FOPT adds fraction of last object.

Fact: $\text{FOPT} \leq \text{GREEDY} + \underbrace{v_{\text{last object}}}_{\leq \max_i v_i}$.

So either:

- GREEDY $\geq$ FOPT/2, or
- $\max_i v_i \geq FOPT/2$.

Algorithm: Max or Greedy by value/size

1. run GREEDY.
2. MAX = $\max_i v_i$.
3. if MAX $\geq$ GREEDY, take MAX
4. else, take GREEDY.

Lemma: alg is 2-approximation.

Proof: ALG $\geq$ FOPT/2 $\geq$ OPT/2.

Pseudo-polynomial Time

"polynomial if numbers in input are written in unary (not binary)"

Integer Knapsack

input:

- n objects $S = \{1, \ldots, n\}$
- s_i = size of object i (integer).
- v_i = value of object i.
- capacity C of knapsack (integer)

output:

- subset $K \subseteq S$ of objects that
  (a) fit in knapsack together (i.e., $\sum_{i \in K} s_i \leq C$)
  (b) maximize total value (i.e., $\sum_{i \in K} v_i$)

Greedy fails, e.g.,

- largest value/size:

  $\mathbf{v} = (C/2 + 2, C/2, C/2)$.

  $\mathbf{s} = (C/2 + 1, C/2, C/2)$.

- smallest value/size:

  $\mathbf{v} = (1, C/2, C/2)$.

  $\mathbf{s} = (2, C/2, C/2)$.

Find a subproblem:

- consider object $i \in S$.
- if i in knapsack:

  value of knapsack is v_i + optimal knapsack value on $S \setminus \{i\}$ with capacity $C - s_i$.

- if i not in knapsack:

  value of knapsack is optimal knapsack on $S \setminus \{i\}$ with capacity C.

Succinct description:

- remaining objects $\{j, \ldots, n\}$ represented by "j"
- remaining capacity represented by $D \in \{0, \ldots, C\}$.

Step I: identify subproblem in English

OPT(j, D)

= "value of optimal size D knapsack on $\{j, \ldots, n\}$"

Step II: write recurrence

OPT(j, D)

$$= \max(\underbrace{v_j + \mathrm{OPT}(j+1, D - s_j)}_{\text{if } s_j \leq D}, \mathrm{OPT}(j+1, D))$$

Step III: base case

OPT$(n + 1, D) = 0$ (for all D)

Step IV: iterative DP

Algorithm: knapsack

1. $\forall D$, memo$[n + 1, D] = 0$.
2. for $i = n$ down to 1,

   for $D = C$ down to 0,

(a) if i fits (i.e., $s_i \leq D$)

memo$[j, D] = \max[\text{OPT}(j+1, D),$

$v_j + \text{OPT}(j+1, D - s_j)]$

(b) else,

memo$[j, D] = \text{OPT}(j+1, D)$

3. return memo$[1, C]$

Correctness

induction

Runtime

$$\begin{aligned} T(n, C) &= O(\# \text{ of subprobs} \times \text{cost per subprob}) \\ &= O(nC). \end{aligned}$$

Note: Knapsack DP is pseudo-polynomial time.