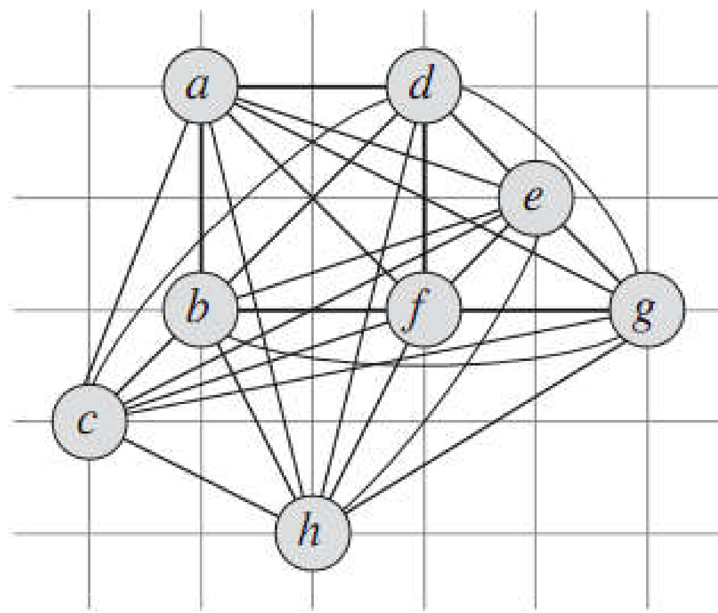
Approximation algorithms 2
TSP, k-Center, Knapsack

CS240 Spring 2024

Rui Fan

Traveling Salesman Problem

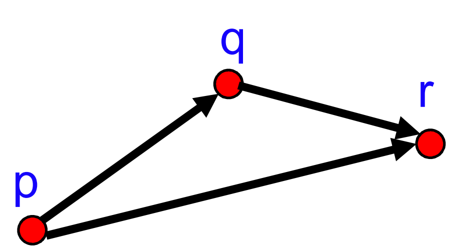
- Input A complete graph with weights on the edges.
- Output A cycle that visits each city once.
- Goal Find a cycle with minimum total weight.



source: CLRS

Metric TSP

- TSP is NP-hard. In fact, it's even NP-hard to approximate when weights can be arbitrary.
- However, TSP is approximable for special types of weights.
- A weighted graph satisfies the triangle inequality if for any 3 vertices p, q, r, we have $d_{pq}+d_{qr} \geq d_{pr}$.
  - I.e., direct path is always no worse than a roundabout path.
  - This is called a metric TSP.
- There is a 1.5-approx algorithm for TSP in graphs with the triangle inequality.
  - Let's look at a simpler 2-approx first.

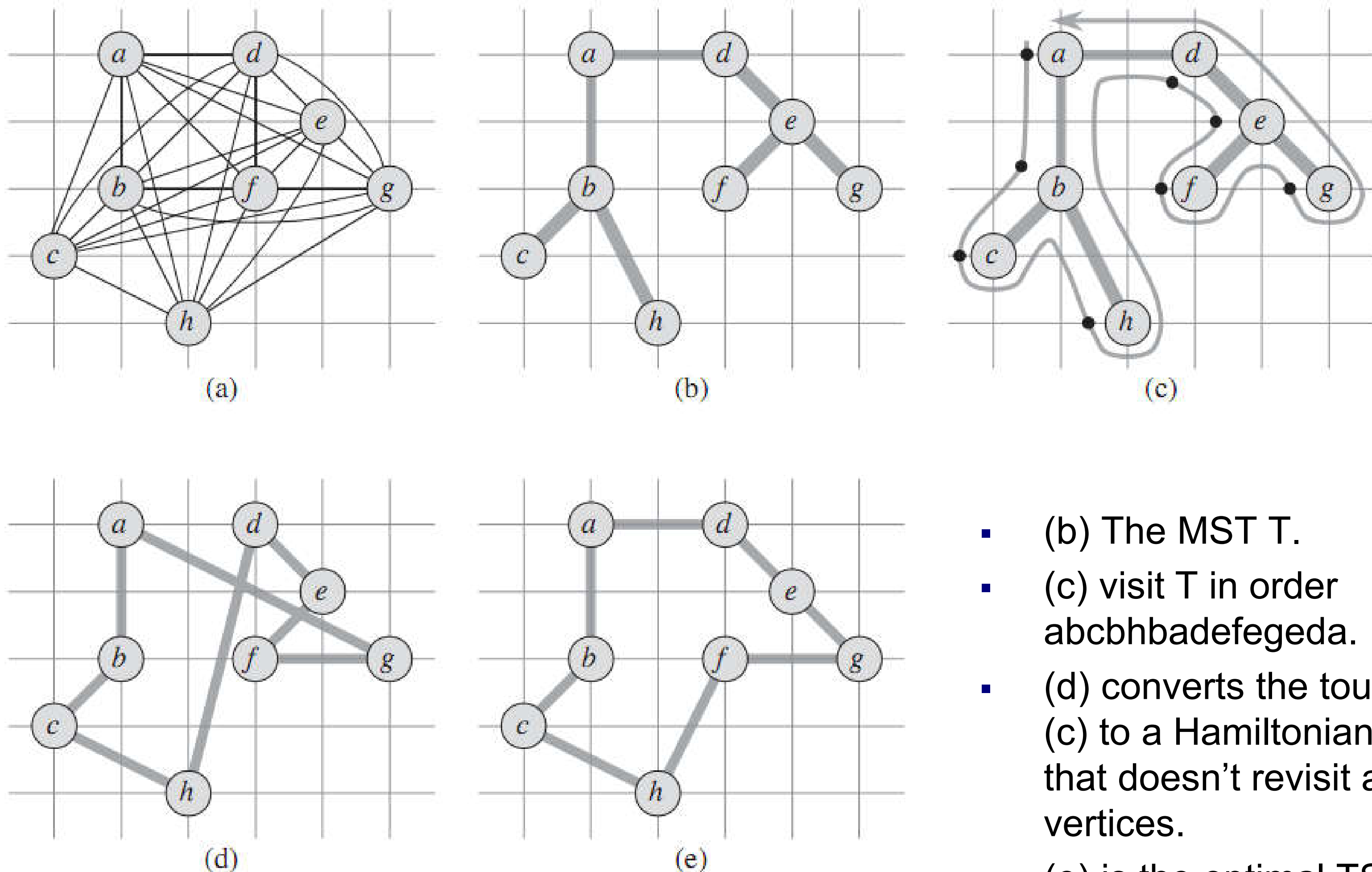


A 2-approximation for TSP

- Construct a minimum spanning tree T on G.
- Use depth-first traversal to visit all the vertices in T, starting from an arbitrary vertex.
- Convert this depth-first traversal T’ to a cycle H that doesn’t revisit any vertex.
- Return H as the TSP tour.

Example

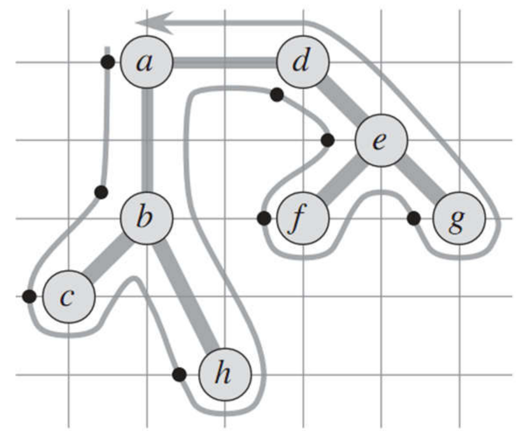
source: CLRS



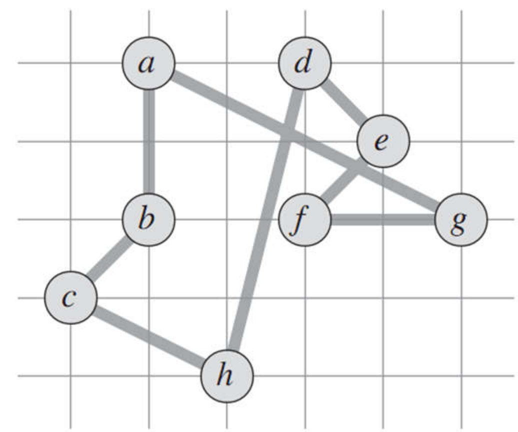
- (b) The MST T.
- (c) visit T in order abcbhbadefegeda.
- (d) converts the tour from (c) to a Hamiltonian cycle, that doesn't revisit any vertices.
- (e) is the optimal TSP.

Making the tour Hamiltonian

- To go from (c) to (d), we need to make a tour T’ that revisits vertices into a cycle H that doesn’t revisit vertices.
- We use shortcutting.
  - If we revisit a vertex in T’, we directly jump to the next vertex in T’ we haven’t visited.
    - We allow revisiting the first vertex.
  - The sequence of vertices we now visit is H.
  - Ex abcbhbadefegeda → abchdefga.



(c)



(d)

Making the tour Hamiltonian

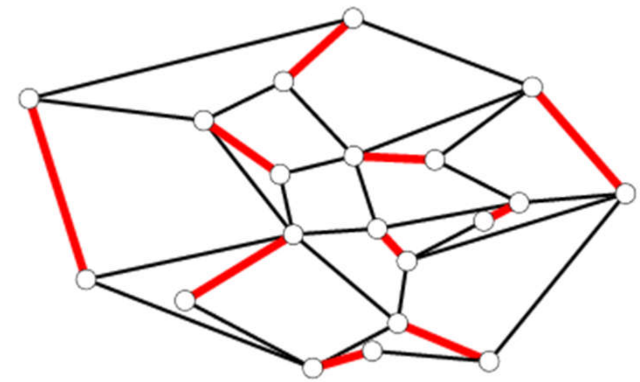
- Lemma If H is the shortcut of T', then c(H)≤c(T').
- Proof We formed H from T' by skipping over some vertices. E.g. we directly went from c to h, skipping over b.
  - But by the triangle inequality, $d_{cb}+d_{bh} \geq d_{ch}$.
    - So shortcutting from c to h didn't increase the distance.
  - The same thing applies to all our shortcuts.
  - So H is no longer than T'.

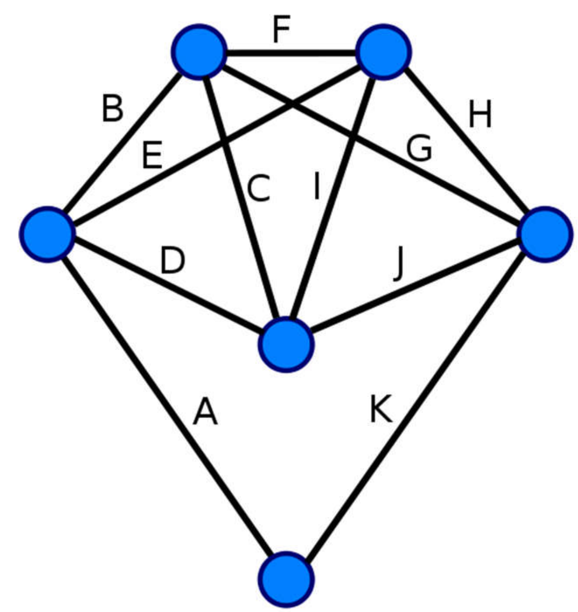
Proof of 2-approximation

- Let H* be an optimum TSP.
- If we delete an edge from H*, we get a spanning tree.
- Since T is an MST, $c(T) \leq c(H^*)$.
- Call the path from the depth-first traversal T'.
  - T' crosses each edge in T twice.
  - So $c(T') = 2\ c(T)$.
- Let H be the outcome of shortcutting T'.
  - H is a Hamiltonian cycle. It visits all the vertices, and ends where it started.
  - $c(H) \leq c(T')$, by the lemma.
  - $c(H) \leq c(T') = 2\ c(T) \leq 2\ c(H^*)$.
- So H is a 2-approximation.

Matchings and Euler cycles

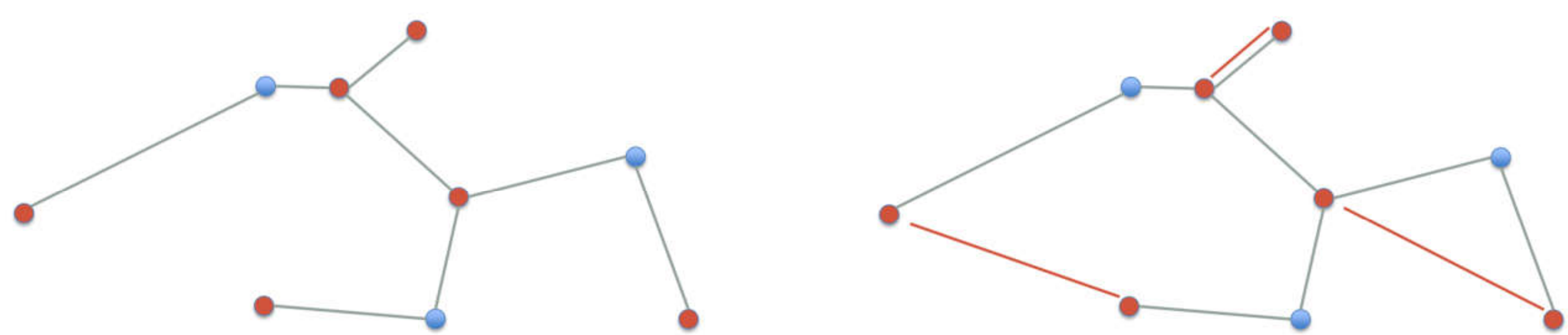
- A matching in a graph is a set of nonintersecting edges.
  - A perfect matching is a matching that includes every vertex.
- An Euler tour of a graph is a path that starts and ends at the same vertex, and visits every edge once.
  - Hamiltonian tour visits every vertex once.
- Thm (Euler) A graph has an Euler tour if and only if all vertices have even degree.
- Note how deciding if graph has Euler tour is trivial, but deciding if it has Hamiltonian tour is NPC!





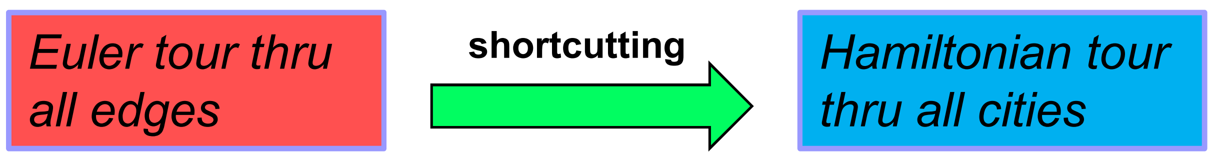
Christofides 3/2-approx algorithm

- A 3/2-approximation for TSP with triangle inequality.
- Construct a minimum spanning tree T on G.
- Find the set V’ of odd degree vertices in T.
- Construct a minimum cost perfect matching M on V’.
- Add M to T to obtain T’.
- Find an Euler tour T’’ in T’.
- Shortcut T’’ to obtain a Hamiltonian cycle H. Output as the TSP.



Why Christofides works well

- In the 2-approx, we found a TSP by “doubling” the MST to an Euler tour, then shortcutting.
  - We need to start with Euler tour before shortcutting to ensure we visit all cities.



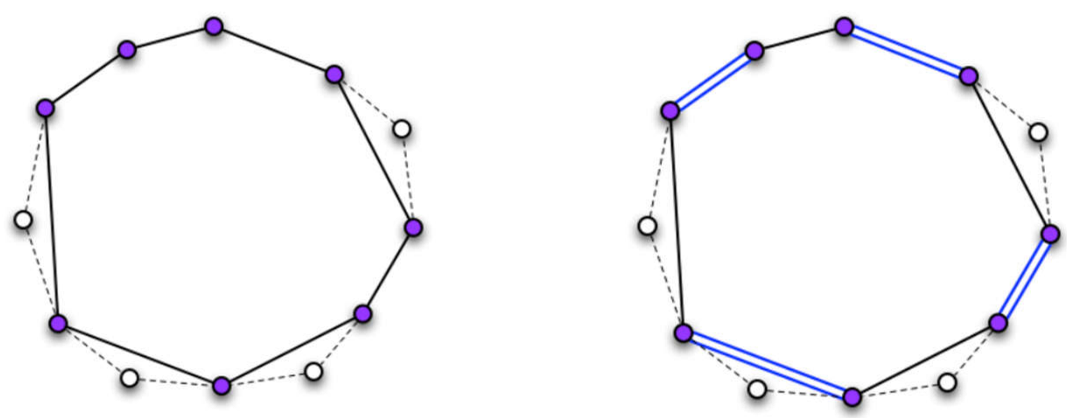
- Key to Christofides is to find a shorter Euler tour, without doubling the MST.
  - A graph with only even degree vertices always has Euler tour.
  - So we want to modify the MST to have all even degrees, by adding a matching.

Proof of correctness

- Lemma T' has an Euler tour.
- Proof There are an even number of vertices in V', because the total degree of T is even.
  - Since G is a complete graph and |V'| is even, there's a perfect matching on V'.
    - The min cost perfect matching can be found in $O(n^2)$ time using the blossom algorithm.
  - The degree of every node in M is odd. Since V' are the odd degree nodes in T, adding M to T makes all nodes in T' have even degree.
  - T' has Euler tour by Euler's theorem.

Proof of correctness

- Lemma Let H* be an optimal TSP on G, and let m be the cost of M. Then $m \leq c(H^*)/2$.
- Proof Let H’ be the optimal TSP on V’.
  - $c(H') \leq c(H^*)$ because H’ is an optimal TSP on fewer vertices.
  - H’ is a cycle on V’, so it consists of two matchings on V’. The cheaper one has cost $m' \leq c(H')/2 \leq c(H^*)/2$.
  - $m \leq m'$ because M has min cost.



http://www.mpi-inf.mpg.de/departments/d1/teaching/ss12/AdvancedGraphAlgorithms/Slides06.pdf

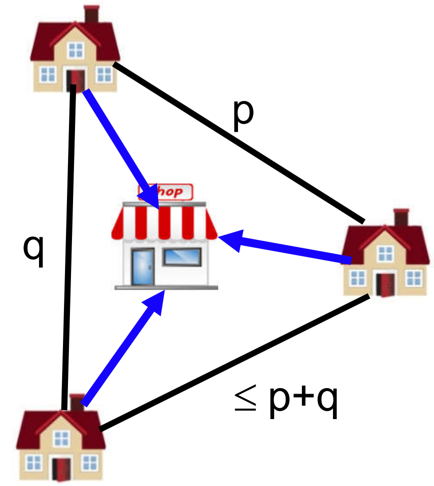
Proof of 3/2-approximation

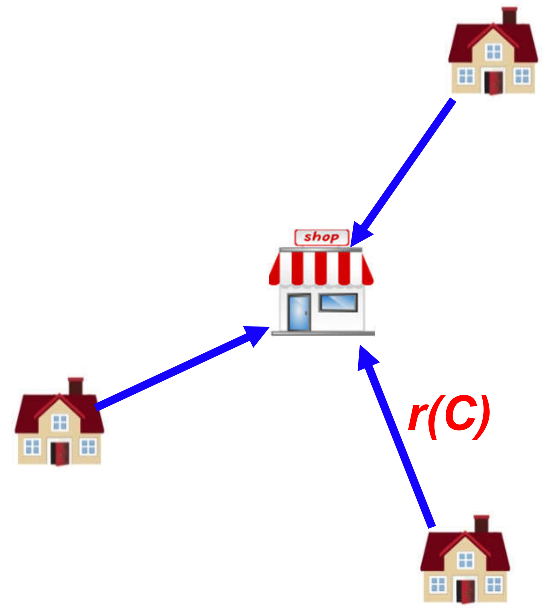
- Thm Let H be the TSP output by Christofides and let H* be an optimal TSP. Then $c(H) \leq 3/2*c(H^*)$.
- Proof
  - $c(T) \leq c(H^*)$ because T is an MST.
  - $c(T') = c(M) + c(T) \leq c(H^*)/2 + c(H^*) = 3/2*c(H^*)$.
  - $c(H) \leq c(T')$ because H is the shortcut of T'.

- *Construct a minimum spanning tree T on G.*
- *Find set V' of odd-degree vertices in T.*
- *Construct a minimum cost perfect matching M on V'.*
- *Add M to T to obtain T'.*
- *Shortcut T' to obtain a Hamiltonian cycle. Output as the TSP.*

k-Center problem

- Given a city with n sites, we want to build k centers to serve them.
  - Let S be set of sites, C be set of centers.
- Each site uses the center closest to it.
  - Distance of site s from the nearest center is $d(s,C) = \min_{c \in C} d(s,c)$.
- Goal is to make sure no site is too far from its center.
  - We want to minimize the max distance that any site is from its closest center.
    - Minimize $r(C)=\max_{s \in S} \min_{c \in C} d(s,c)$.
  - C is called a cover of S, and r is called C's radius.
  - Where should we put centers to minimize the radius?
- Assume distances satisfy triangle inequality.





Gonzalez's algorithm

- k-Center is NP-complete.
- We'll give a simple 2-approximation for it.
- Idea Say there's one site that's farthest away from all centers. Then it makes the radius large. We'll put a center at that site, to reduce the radius.
  - Note we allow putting center at same location as site.

Gonzalez's algorithm

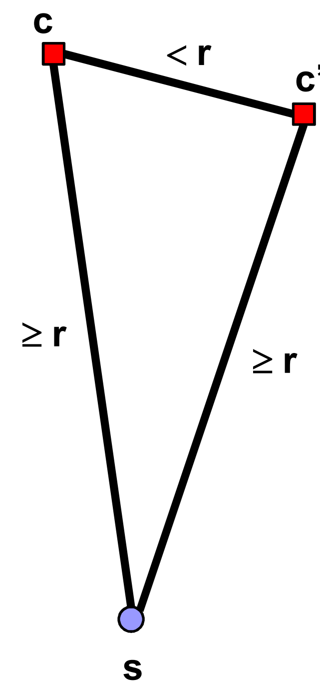
- C is set of centers, initially empty.

- repeat k times
  - choose site s with maximum d(s,C)
  - add s to C
- return C

- Note The centers are located at the sites.

Proof of correctness

- Let C be the algorithm's output, and r be C's radius.
  - $r = \max_{s \in S} \min_{c \in C} d(s,c)$
- Lemma 1 For any $c,c' \in C$, $d(c,c') \geq r$.
- Proof Since r is the radius, there exists a point $s \in S$ at distance $\geq r$ from all the centers.
  - If there's no such s, then C's radius < r.
  - So s is distance $\geq r$ from c and c'.
  - Suppose WLOG c' is added to C after c.
  - If $d(c,c')<r$, then algorithm would add s to C instead of c', since s is farther.

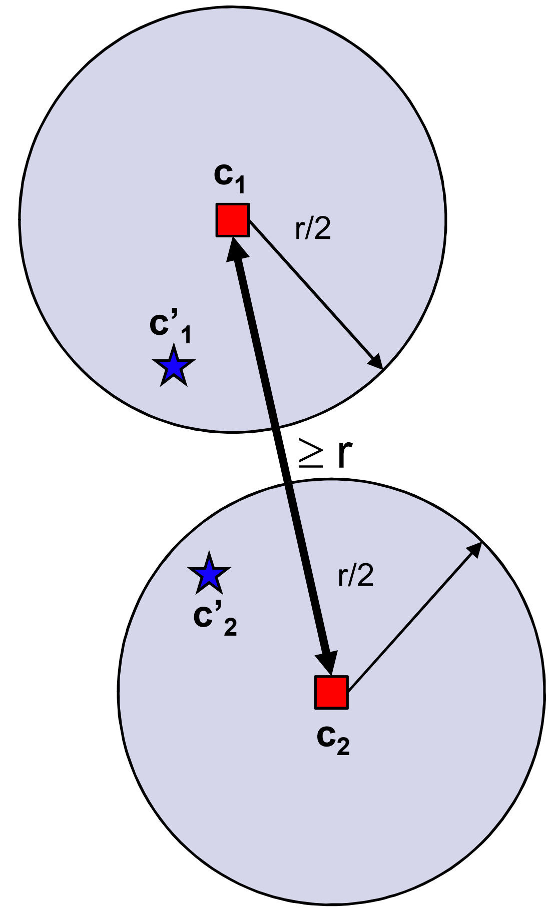


Proof of correctness

- Cor There exist k+1 points mutually at distance $\geq$ r from each other.
  - By the lemma, the k centers are mutually $\geq$ r distance apart.
  - Also, there's an $s \in S$ at distance $\geq$ r from all the centers.
    - Otherwise C's covering radius is $< r$.
  - So the k centers plus s are the k+1 points.
- Call these k+1 points D.

Proof of correctness

- Let C^* be an optimal cover with radius r^*.
- Lemma 2 Suppose $r > 2r^*$. Then for every $c \in D$, there exists a corresponding $c' \in C^*$. Furthermore, all these c' are unique.
- Proof Draw a circle of radius $r/2$ around each $c \in D$.
  - There must be a $c' \in C^*$ inside the circle, because
    - c is at most distance r^* away from its nearest center, since r^* is C^*'s radius.
    - $r/2 > r^*$.
  - Given $c_1, c_2 \in D$, let $c'_1, c'_2 \in C^*$ be inside c_1 and c_2's circle, resp.
  - c_1 and c_2's circles don't touch, because $d(c_1, c_2) \geq r$.
  - So $c'_1 \neq c'_2$.

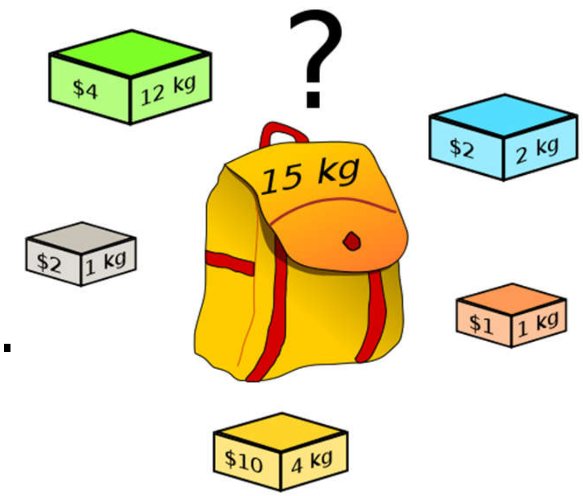


Proof of correctness

- Thm Let C be the output of Gonzalez's algorithm, and let C* be an optimal k-center. Then $r(C) \leq 2r(C^*)$.
- Proof By Lemma 2, if $r(C)>2r(C^*)$, then for every $c \in D$, there is a unique $c' \in C^*$.
  - But there are k+1 points in D, by the corollary.
  - So there are k+1 points in C*. This is a contradiction because C* is a k-center.

The knapsack problem

- We have a set of items, each having a weight and a value.
- We have a knapsack that can carry up to W amount of weight.
- We want to put items in the knapsack to maximize the total value, but not exceed the weight limit.
- Ex Items 3 and 4 are the highest value items with weight ≤ 11.
- Assume all items have weight $\leq$ W, i.e. any single item fits in knapsack.



W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

A dynamic program for knapsack

- Let OPT(i,v) = minimum weight of a subset of items 1,...,i that has value $\geq$ v.
- If optimal solution uses item i.
  - Then we pay w_i weight for item i, and need to achieve value $\geq v-v_i$ using items 1,...,i-1 using min weight.
  - So $OPT(i,v)=w_i+OPT(i-1,v-v_i)$.
- If optimal solution doesn't use item i.
  - Then we need to achieve value $\geq$ v using items 1,...,i-1.
  - So OPT(i,v)=OPT(i-1,v).
- Choose the case that gives smaller weight.
- $$OPT(i,v) = \begin{cases} 0 & \text{if } v=0 \\ \infty & \text{if } i=0,\ v>0 \\ \min(OPT(i-1,v),\ w_i+OPT(i-1,v-v_i)) & \text{otherwise} \end{cases}$$

Running time of dynamic program

- Say there are n items, and the largest value of any item is v^*.
- The max value we can pack into the knapsack is nv^*, where v^* is the largest v value.
- Solve all subproblems of the form OPT(i,v), where $i \leq n$ and $v \leq nv^*$.
  - This is a total of $O(n^2v^*)$ subproblems.
- The solution to Knapsack is the max value V that can be packed with weight $\leq W$.
- Having solved all the subproblems, we can find V by finding the subproblem with the largest value that has optimum weight $\leq W$.
  - $V = \max_{v \leq nv^*} OPT(n,v) \leq W$.
- So solving Knapsack takes total time $O(n^2v^*)$.

Running time of dynamic program

- The DP gives an optimal solution to Knapsack and takes $O(n^2 v^*)$ time. Have we found a polytime algorithm for an NP-complete problem?
- No. The problem size is $O(n \log(v^*))$, because it takes $\log(v^*)$ bits to express each item's value. But $O(n^2 v^*)$ is not polynomial in $n \log(v^*)$.
- To make this DP fast, we have to make the largest value small.

PTAS

- Let $\varepsilon>0$ be any number. We'll give a $(1+\varepsilon)$-approximation for knapsack.
- By setting ε sufficiently small, we can get as good an approximation as we want!
  - This type of algorithm is called a polynomial time approximation scheme, or PTAS.
- Contrast this with earlier algs we studied, which had worse approx ratios, e.g. 2 or log n.
- But the running time will be $O(n^3/\varepsilon)$. Hence we can't set $\varepsilon=0$ get the optimal solution.
- We're trading accuracy for time. The more accurate (smaller ε), the more time the algorithm takes.

Main idea: rounding

- Since we only need an approximate solution, we can change the values of the items a little (round the values) and not affect the solution much.
- We scale and round the original values to make them small.
- The previous DP took $O(n^2v^*)$ time. So if the rounded values are small, this DP is fast.

W = 11

Item	Value	Weight
1	134,221	1
2	656,342	2
3	1,810,013	5
4	22,217,800	6
5	28,343,199	7

→

W = 11

Item	Value	Weight
1	2	1
2	7	2
3	19	5
4	223	6
5	284	7

Rounding

- Let $\varepsilon>0$ be the precision we want.
- Set $\theta = \frac{\epsilon v^*}{2n}$ to be a scaling factor.
  - v^* is the largest value of any item.
- Scale all values down by θ then round up.
  - $v' = \lceil v/\theta \rceil$.
- Make a problem where each value v_i is replaced by v'_i.
  - Call this the scaled rounded problem.
- Let v^ be max value in the scaled rounded problem. Then $\hat{v} = \lceil v^*/\theta \rceil = \lceil v^*/(\varepsilon v^*/2n) \rceil = \lceil 2n/\varepsilon \rceil$.
- Running time of DP on scaled rounded problem is $O(n^2 \hat{v}) = O(n^3/\varepsilon)$.

Solving the original problem

- Make another new problem in which each value v_i is replaced by $u_i = \lceil v_i/\theta \rceil * \theta$.
  - Call this the rounded problem.
  - We have $u_i \geq v_i$, and $u_i \leq v_i + \theta$.
- Note u values are equal to v' values multiplied by θ.
  - Thus, the optimal solution for the rounded problem and the scaled rounded problem are the same.
- We now have 3 problems, the original problem, the scaled rounded problem, and the rounded problem.
- Let S be the optimal solution to the scaled rounded problem, which we can find in time $O(n^3/\varepsilon)$. S is also optimal for the rounded problem.
- We'll show S is a $1+\varepsilon$ approximation for the original problem.

Correctness

- Thm Let S* be the optimal solution to the original problem. Then $(1+\varepsilon)\sum_{i \in S} v_i \geq \sum_{i \in S^*} v_i$. Hence S is a (1+ε)-approximate solution.

- Proof

$\sum_{i \in S^*} v_i \leq \sum_{i \in S^*} u_i$ $\qquad$ $u_i \geq v_i$

$\leq \sum_{i \in S} u_i$ $\qquad$ S is opt soln for rounded problem

$\leq \sum_{i \in S} (v_i + \theta)$ $\qquad$ $u_i \leq v_i + \theta$

$\leq \sum_{i \in S} v_i + n\theta$ $\qquad$ $|S| \leq n$

Correctness

- Suppose item j has the largest value, so $v^*=v_j$. Then $n\theta = \frac{\varepsilon}{2} v_j \le \frac{\varepsilon}{2} u_j \le \frac{\varepsilon}{2} \sum_{i \in S} u_i$
  - Last inequality because item j itself is feasible solution, so opt solution S is no smaller.
- So $\sum_{i \in S} v_i \ge \sum_{i \in S} u_i - n\theta \ge \left(\frac{2}{\varepsilon} - 1 \right) n\theta$, where first inequality comes inequalities on last page.
- Assuming $\varepsilon \le 1$, then $n\theta \le \varepsilon \sum_{i \in S} v_i$
- Finally, we have

$$\sum_{i \in S^*} v_i \le \sum_{i \in S} v_i + n\theta \le \sum_{i \in S} v_i + \varepsilon \sum_{i \in S} v_i = (1+\varepsilon) \sum_{i \in S} v_i$$

Summary

- We gave a DP for Knapsack.
- We scale and round to reduce number of different item values.
- Running the DP on the scaled rounded problem and using the solution for the original problem leads to an arbitrarily good approximation for Knapsack, a PTAS.
- There are PTAS's for a number of other problems.
  - Multiprocessor scheduling.
  - Bin packing.
  - Euclidean TSP.
- However, there are also many problems for which PTAS's do not exist, unless P=NP.