Lec. 11: Approximation Algorithms

Outline

- Approximation Algorithms

- Load Balancing
- Bin Packing
- Center Selection

- Maximum Weighted Cut
- Maximum Coverage

- Weighted Vertex Cover

- Metric Travelling Salesman Problem

- Knapsack Problem

NP-hard optimization problems

Q. Suppose I need to solve an NP-hard problem. What should I do?
A. Theory says you're unlikely to find a poly-time algorithm.

Must sacrifice one of three desired features.

- Solve problem to optimality.
- Solve problem in poly-time.
- Solve arbitrary instances of the problem.

Approximation algorithms

Key: provably close to optimal.

OPT: the value of an optimal solution,
SOL: the value of the solution that our algorithm returned.

Additive approximation algorithms:
- $SOL \leq OPT + c$ for a minimization problem
- $SOL \geq OPT - c$ for a maximization proble

Multiplicative approximation algorithms:
- $SOL \leq c \cdot OPT$ for a minimization problem
- $SOL \geq OPT/c$ for a maximization problem

Challenge. Need to prove a solution's value is close to optimum, without even knowing what optimum value is!

Time-accuracy tradeoff

Def. An algorithm ***A*** is a **PTAS** (Polynomial-Time Approximation Scheme) if for every $\epsilon > 0$, ***A*** runs in polynomial time (which may depend on ϵ) and return a $(1+\epsilon)$ -approximate solution

- For example, ***A*** may run in time $n^{100/\epsilon}$.

Def. An algorithm ***A*** is a **FPTAS** (fully PTAS) if for every $\epsilon > 0$, ***A*** runs in time poly(n, $1/\epsilon$) and return a $(1+\epsilon)$ -approximate solution

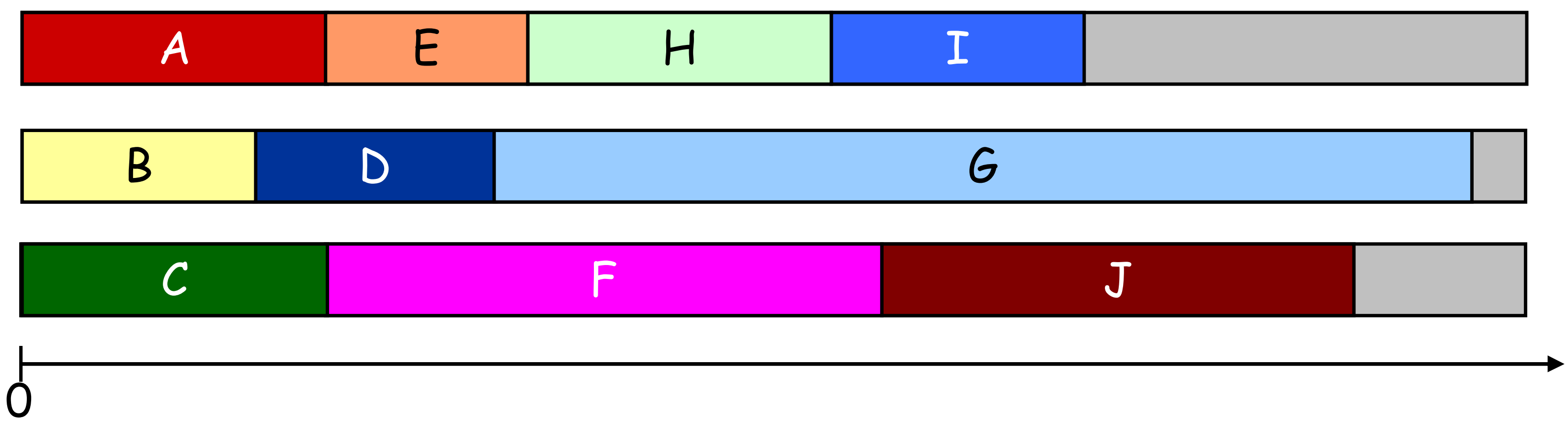
1. Load Balancing

Load Balancing

Input. m identical machines; $n > m$ jobs, job j has processing time t_j.

Job scheduling:

- Each job must run contiguously on one machine.
- A machine can process at most one job at a time.



load of machine i is L_i = sum process times of jobs assigned to machine i

makespan $L = \max_i L_i$.

Load balancing. Assign each job to a machine to minimize makespan.

List scheduling algorithm

Consider n jobs in some fixed order (i.e. list).
Assign job j to machine whose load is smallest so far.

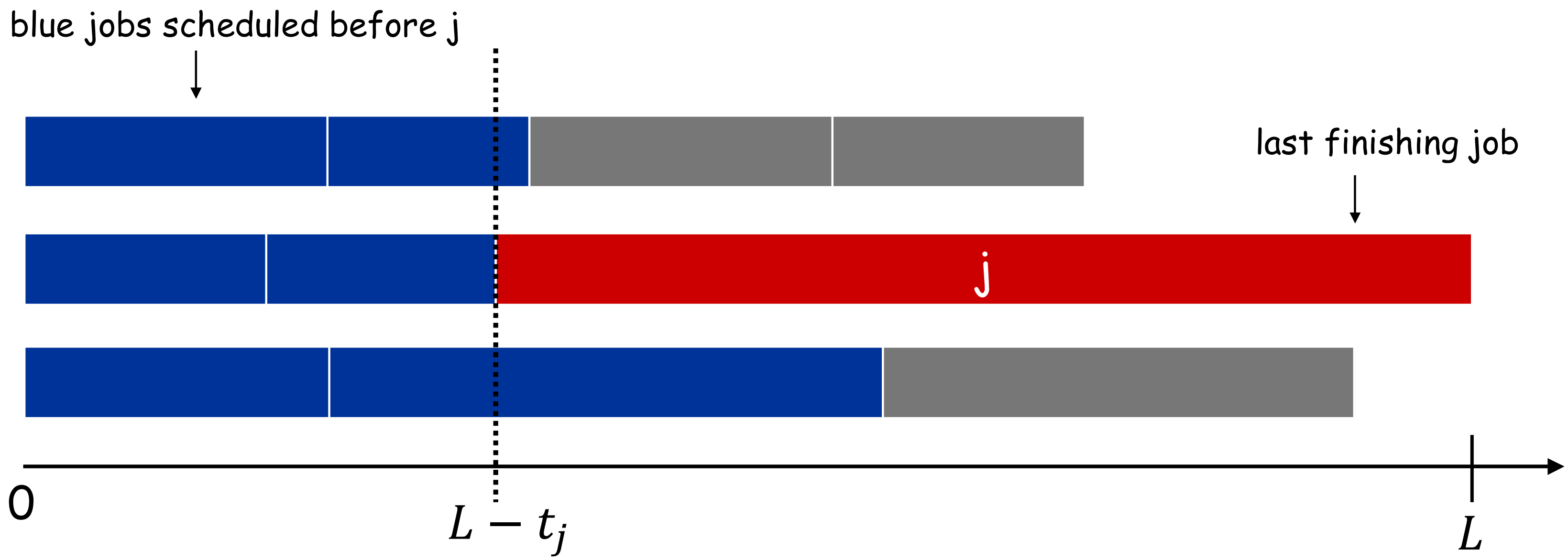
Thm. [Graham 1966] LS algorithm is a 2-approximation.

- First worst-case analysis of an approximation algorithm.
- Need to compare resulting solution with optimal makespan L^*.

Lem. $L^* \geq \max\{\max_{1\leq j\leq n} t_j, \frac{1}{m}\sum_{j=1}^{n} t_j\}$.

List scheduling analysis

Thm. LS algorithm is a 2-approximation.



Pf. j's starting time $L - t_j$ = least load of machines at this time

$$\le \frac{1}{m}\sum_{i=1}^{j-1} t_i \le L^* - \frac{1}{m} t_j$$

Thus,

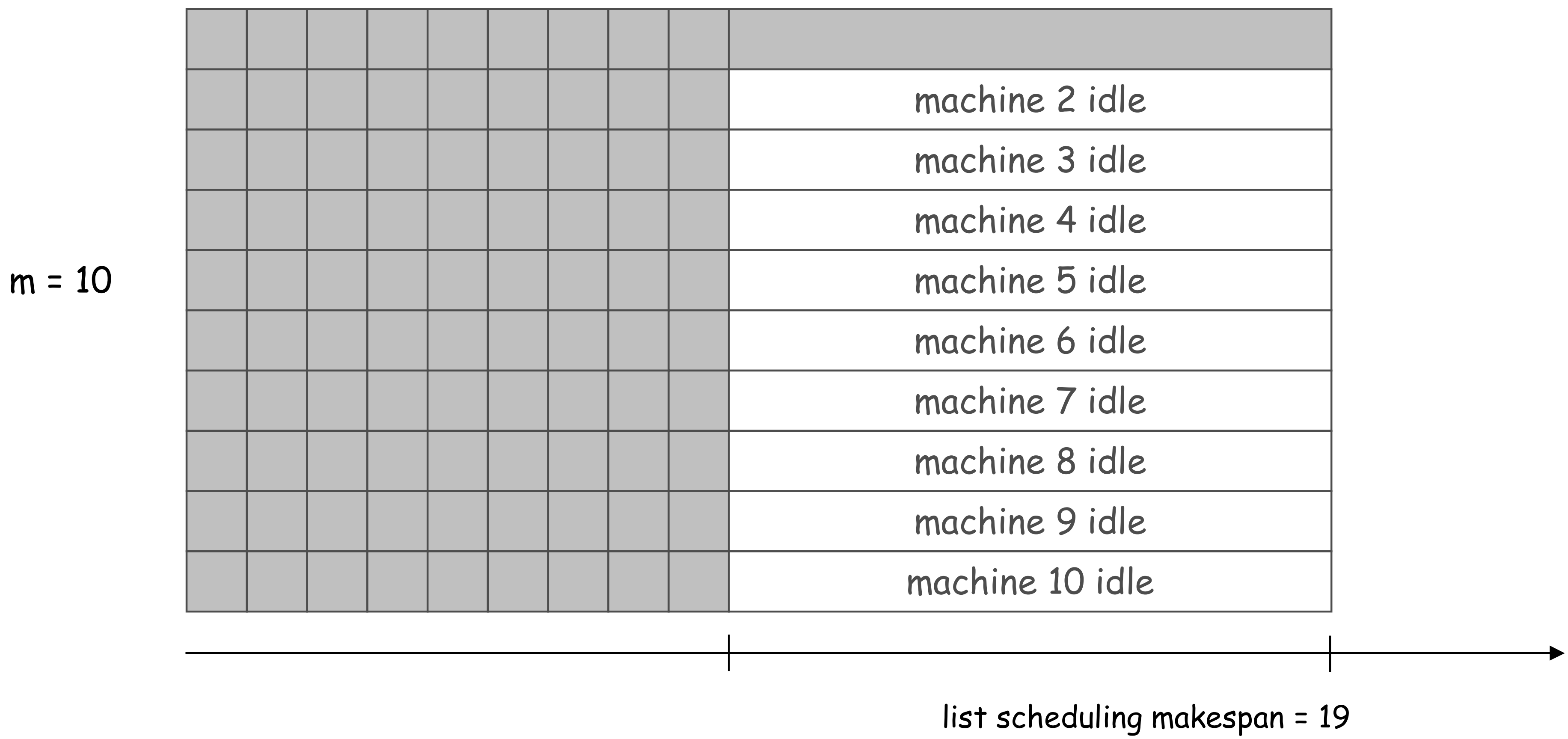
$$L \le L^* + \left(1 - \frac{1}{m}\right) t_j \le L^* + \left(1 - \frac{1}{m}\right) L^* = \left(2 - \frac{1}{m}\right) L^*$$

Tight instance

Q. Is our analysis tight?

A. Essentially yes.

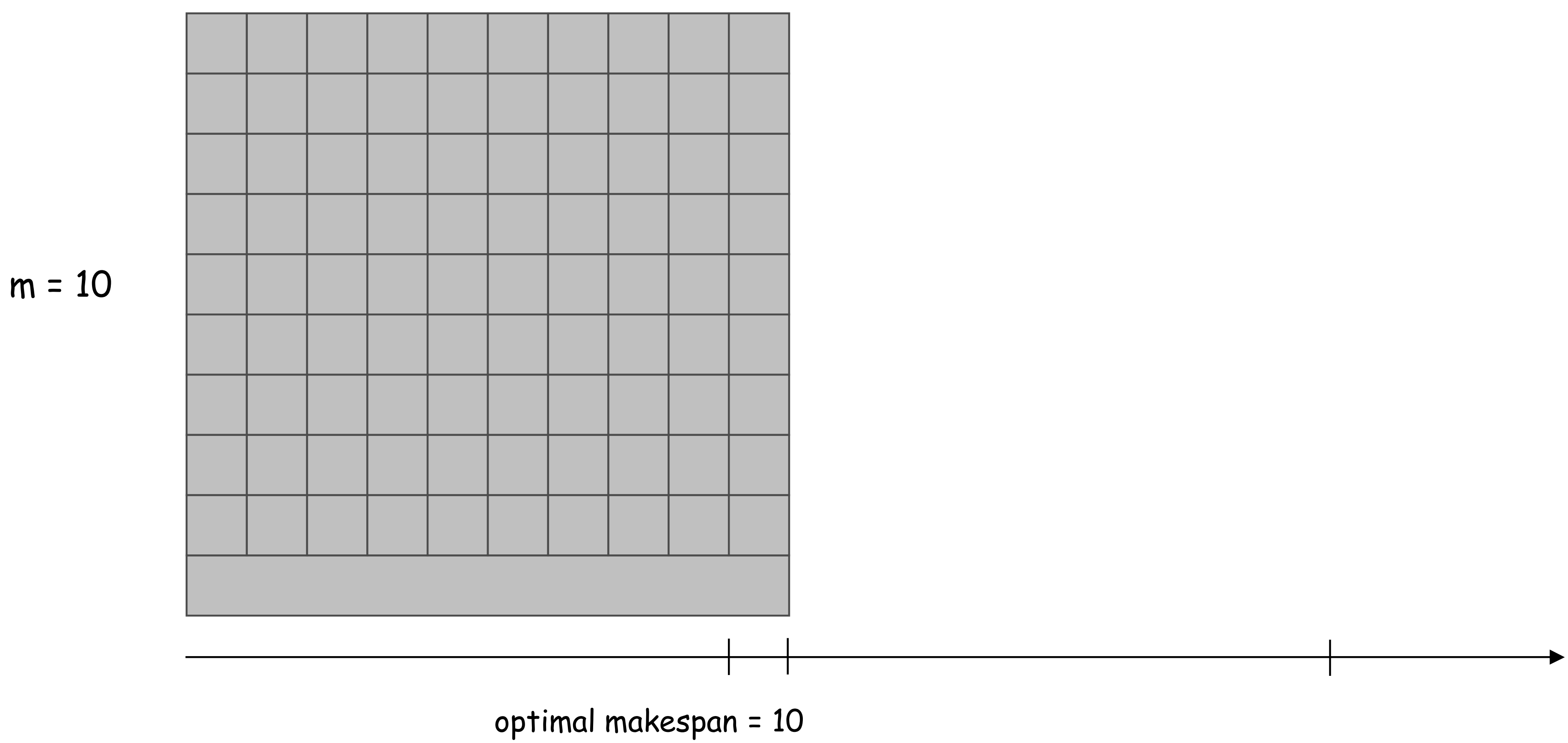
Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m



Tight instance

Q. Is our analysis tight?
A. Essentially yes.

Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m



LPT Rule

Rationale: $L \leq L^* + \left(1 - \frac{1}{m}\right) t_j$

- place the shorter jobs more towards the end of the schedule, where they can be used for finer load balancing.

Longest processing time (LPT). Sort n jobs in descending order of processing time, and then run list scheduling algorithm.

Observation. The first m jobs are put on m different machines

LPT Rule: simple analysis

Lem. $L^* \geq 2t_{m+1}$.

Pf. Among the first $m+1$ jobs, at least two are in the same machine ▪

Thm. LPT rule is a 3/2 approximation algorithm.

Pf. If the last-finishing job $j \leq m$, then $L = L^*$.

Otherwise

$$L \leq L^* + \left(1 - \frac{1}{m}\right) t_j \leq L^* + \frac{1}{2}\left(1 - \frac{1}{m}\right) L^* = \left(\frac{3}{2} - \frac{1}{m}\right) L^*$$

LPT Rule: tighter analysis

Q. Is our 3/2 analysis tight?
A. No.

Thm. [Graham 1969] LPT rule is a 4/3-approximation.
Pf. exercise.

Q. Is Graham's 4/3 analysis tight?
A. Essentially yes.

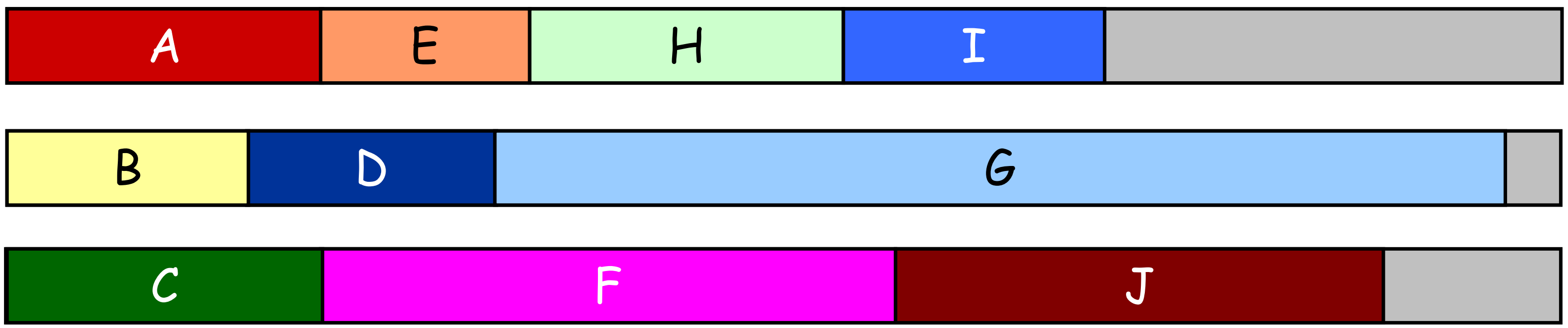
Tight instance: exercise

2. Bin Packing

Bin Packing

Input: n jobs, job j has processing time $t_j \leq 1$.

Packing: A partition of the jobs into groups of total load ≤ 1. Each group corresponds to a bin (machine)



Bin packing: Find a partition with fewest groups (bins).

NP-complete to ***approximate*** within a factor less than 3/2!

Lower bound on optimum

B^*: minimum number of bins

L: the set of "long" jobs with processing time $> 1/2$

Observation. $B^* \geq \max\{\lceil \sum_j t_j \rceil, |L|\}$.

First-Fit (FF) algorithm

Put items in some fixed list (order), and for each job in the list:

- If the job fits into one of the currently open bins, then put it in the first of these bins.
- Otherwise, open a new bin and put the new job in it.

Key property. Among all open bins, all but one are more than half-full.

Thm. FF algorithm is a 2-approximation.

Pf. If $B > 2B^*$, then total load $> (2B^*)/2 \ = B^* \geq \lceil \sum_j t_j \rceil$.

First-Fit Decreasing (FFD) algorithm

FFD: Sort jobs in decreasing order of t_j, and then run FF algorithm.

Thm. FFD algorithm is a 3/2-approximation.

Pf. $k := \left\lceil \frac{2}{3} B \right\rceil$ and it is sufficient to show $k \le \max\{|L|, \lceil \sum_j t_j \rceil\}$.
Assume $k > |L|$, and $S :=$jobs in the bins $k, k+1, \cdots, B$.

- Each job in S is short and doesn't fit any of the first $k-1$ bins.
- $|S| \ge 2(B-k) + 1 \ge k - 1$

Pair up $k-1$ jobs in S with the first $k-1$ bins.
Each pair has total load > 1, and the total load of all pairs $> k-1$.

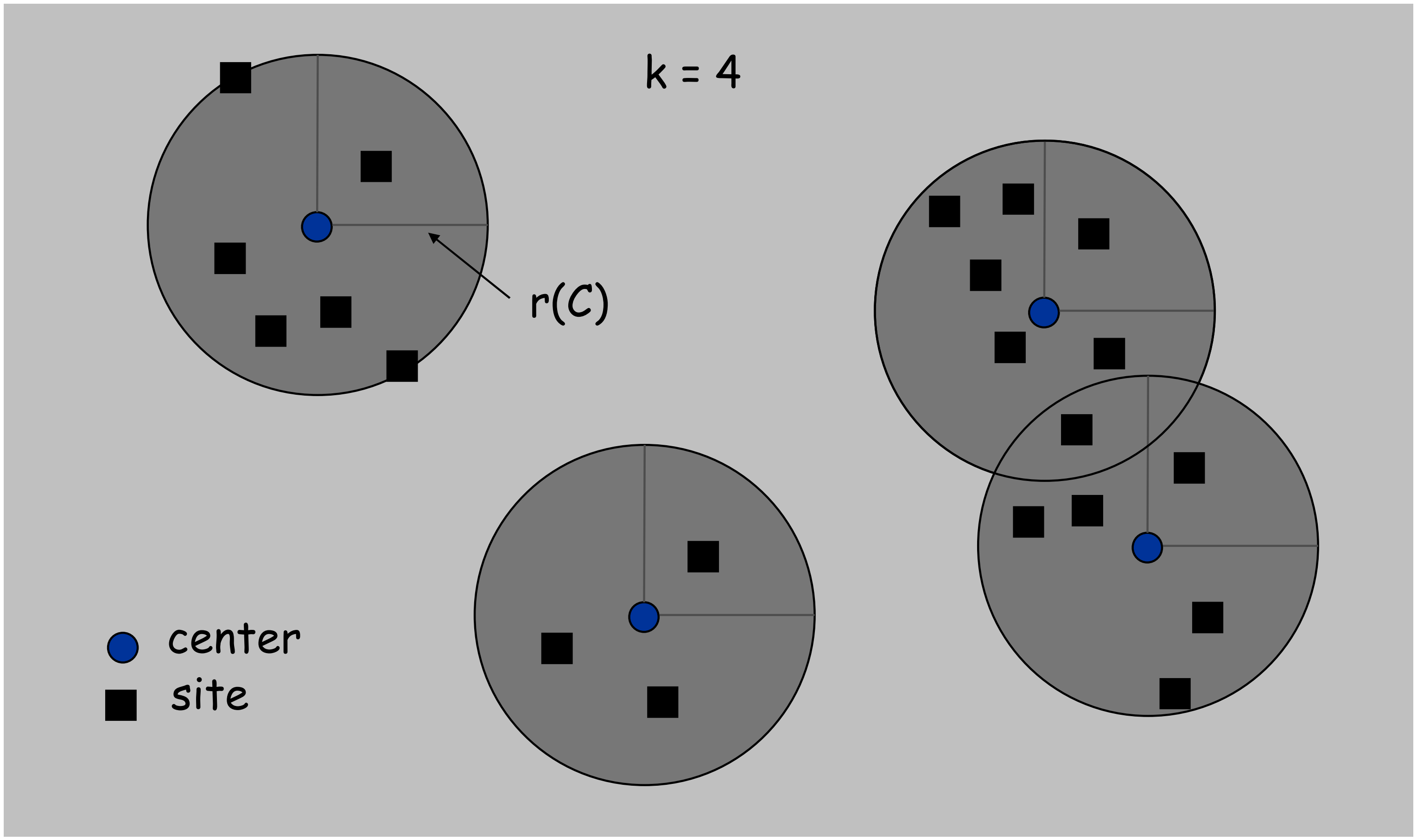
$$k - 1 < \sum_j t_j \Rightarrow k \le \left\lceil \sum_j t_j \right\rceil$$

3. Center Selection

Center Selection Problem

Input. n sites $s_1, \ldots, s_n$.

Center selection problem. Select k centers C so that maximum distance from a site to nearest center is minimized.



Center Selection Problem: metric distances

Notation.

- $dist(x, y)$ = distance between x and y.
- $dist(s_i, C) = \min_{c \in C} dist(s_i, c)$ = distance from s_i to closest center.
- $r(C) = \max_i dist(s_i, C)$ = smallest covering radius.

Goal. Find set of centers C that minimizes $r(C)$, subject to $|C| = k$.

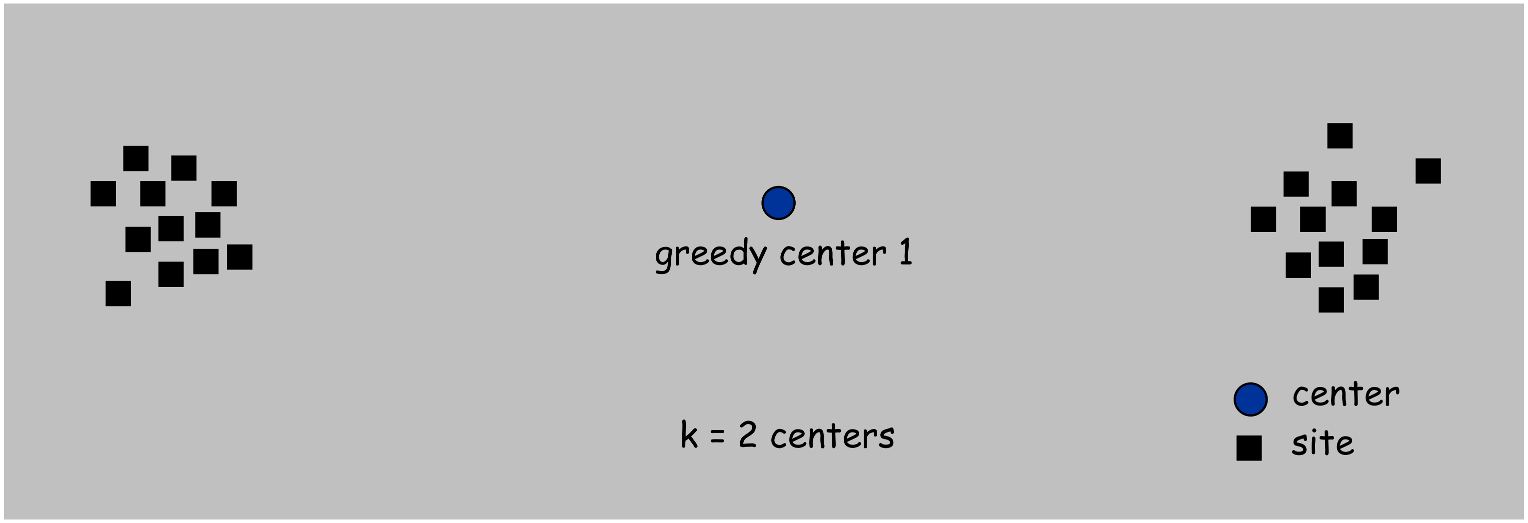
Metric Distance:

- $dist(x, x) = 0$ (identity)
- $dist(x, y) = dist(y, x)$ (symmetry)
- $dist(x, y) \leq dist(x, z) + dist(z, y)$ (triangle inequality)

Greedy algorithm: a false start

- Put the first center at the best possible location for a single center
- Keep adding centers so as to reduce the covering radius each time by as much as possible.

Remark: arbitrarily bad!



Greedy algorithm

- Place the first center at an arbitrary given site, and
- repeatedly place the next center at the (the most dissatisfied) site farthest from any existing center.

```
C = {s_1}
repeat k-1 times
   Select a site s_i with maximum dist(s_i,C)
   Add s_i to C
                         site farthest from any center
return C
```

Observation. All centers in C are pairwise at least $r(C)$ apart.

Analysis of greedy algorithm

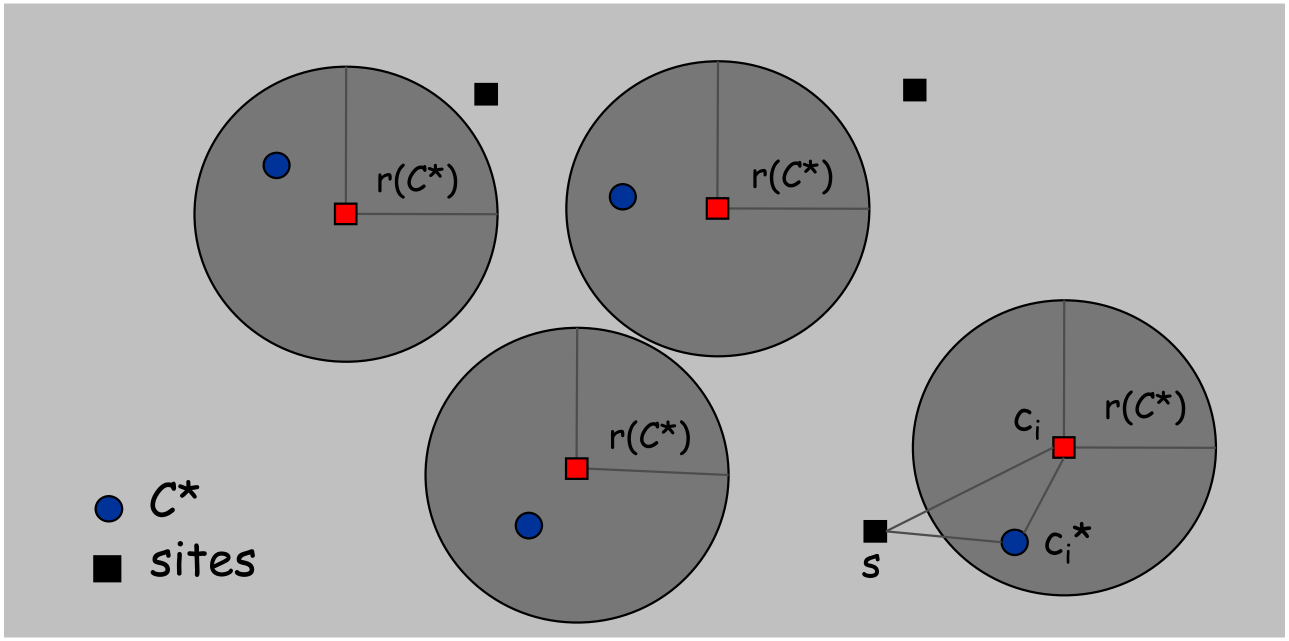
Thm. Let C^* be an optimal set of k centers. Then $r(C) \leq 2r(C^*)$.

Pf. (by contradiction) Assume $r(C^*) < \frac{1}{2} r(C)$.

- The k disks centered at C of radius $r(C^*)$ are disjoint.
- Exactly one optimal center c_i^* in each disk;
- Let c_i be the center of the disk containing c_i^*.
- For any site s and its closest center c_i^* in C^*,

  $$\text{dist}(s, C) \leq \text{dist}(s, c_i) \leq \text{dist}(s, c_i^*) + \text{dist}(c_i^*, c_i) \leq 2r(C^*).$$

- Thus $r(C) \leq 2r(C^*)$. ▪



Approximation hardness

Thm. Greedy algorithm is a 2-approximation for center selection problem.

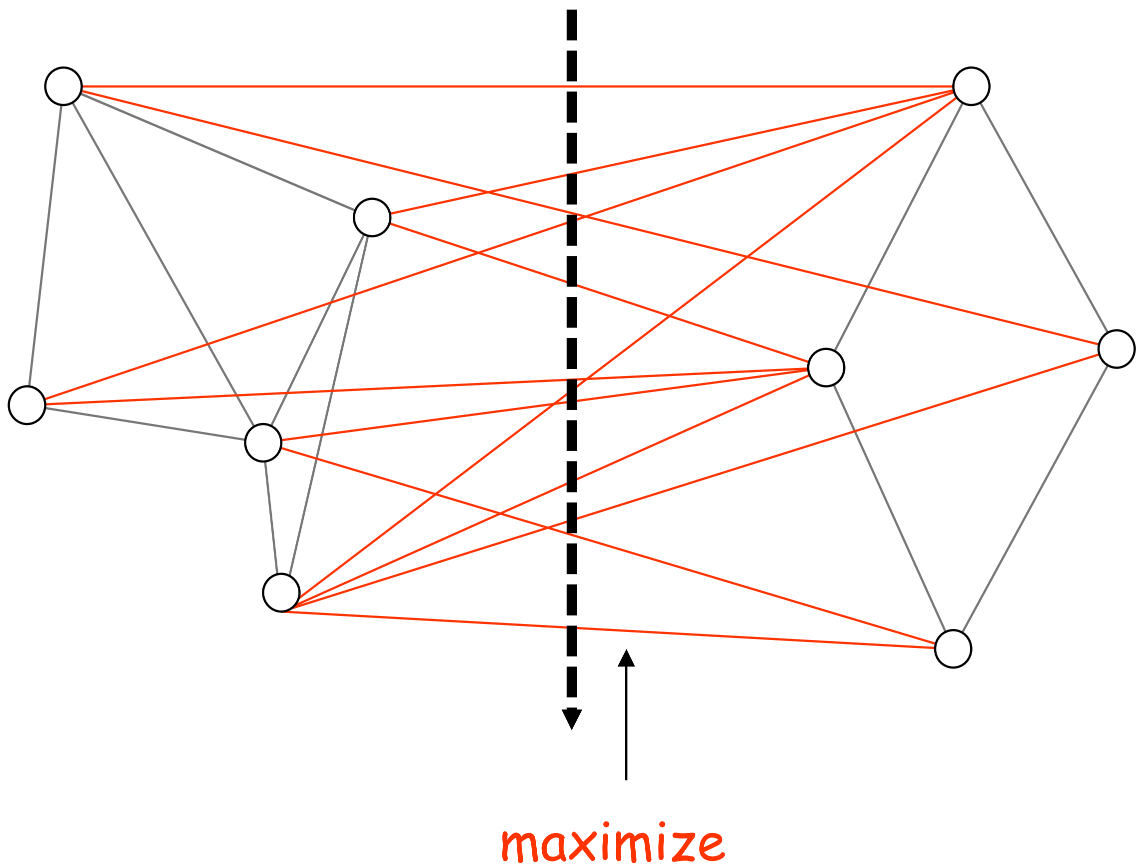
Q. Is there hope of a 3/2-approximation? 4/3?

Thm. Unless P = NP, there is no ρ-approximation for center-selection problem for any $\rho < 2$.

4. Maximum Weighted Cut

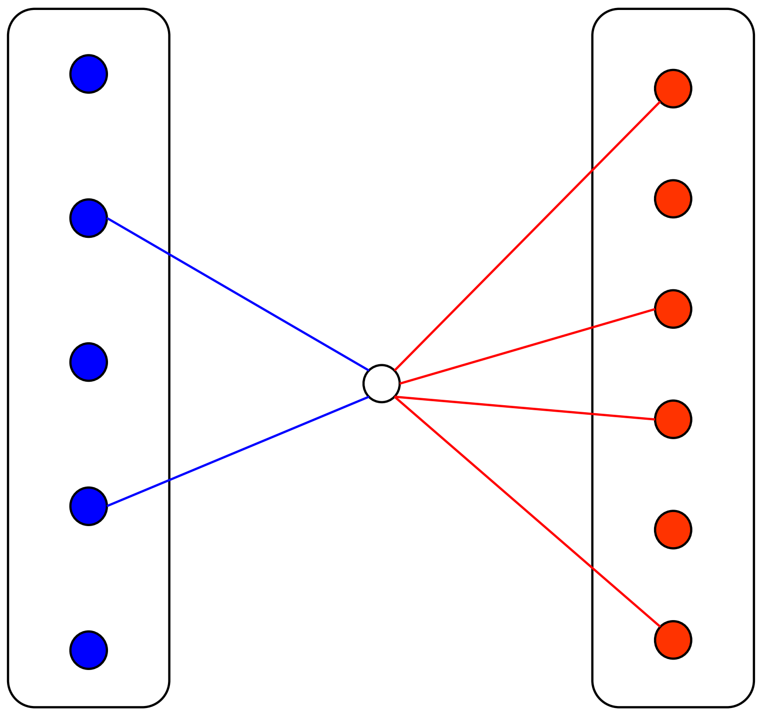
Maximum Weighted Cut

Input: a non-negative edge-weighted graph $G = (V, E; w)$
Max-weighted cut problem: find a cut of maximum total weight



Greedy algorithm

- Pick an arbitrary ordering $1, 2, \ldots, n$ of nodes
- Place node 1 at the left side, and repeatedly place the next node at the better side.



Analysis of greedy algorithm

Thm. The weight of the greedy cut is at least $w(E)/2$. Hence the greedy algorithm is 2-approximate.

Pf.

- Partition E into $E_2, E_3, \dots, E_n$, where
  E_i = the set of edges between i and $\{1, \dots, i-1\}$.
- Node i contributes at least $w(E_i)/2$ towards the cut.
- The total weight of the cut is at least $w(E)/2$.

Better approximation

- (1/0.878)-approximation: [Goemans-Williamson *1995]* using semidefinite programming
  - best possible if the Unique Games Conjecture is true

- NP-hard to approximate better than 17/16 [Håstad 2001]

5. Maximum Coverage

Maximum Coverage

Input: a bipartite graph $G = (U, W; E)$, and a positive integer $k \leq |U|$
Goal: Find a k-subset S of U maximizing $|N(S)|$

```
S := ∅;
repeat k times
    u := a node in U with maximum degree in G;
    add u to S and remove u and its neighbors from G;
return S
```

Thm. Let S^* be an optimal solution and $opt \coloneqq |N(S^*)|$. Then

$$|N(S)| \geq [1 - (1 - 1/k)^k]opt \geq (1 - 1/e)opt.$$

Geometric decreasing of the optimality gap

S: $u_1, u_2, \cdots, u_k$

$n_0 \coloneqq 0$; and $n_i \coloneqq |N(\{u_1, \cdots, u_i\})|$ for $1 \le i \le k$

Claim. $(opt - n_i) \le (1 - 1/k)(opt - n_{i-1})$.

Pf. At the beginning of the iteration i,

- # of nodes in $N(S^*)$ remaining in $G \ge opt - n_{i-1}$
- $n_i - n_{i-1} =$ max degree of $G \ge (opt - n_{i-1})/k$

Thus,

$$(opt - n_{i-1}) - (opt - n_i) = n_i - n_{i-1} \ge (opt - n_{i-1})/k$$

Approximation bound

$$(opt - n_k) \leq (1 - 1/k)^k opt$$
$$|N(S)| = n_k \geq [1 - (1 - 1/k)^k] opt$$

Q: Can we do better?

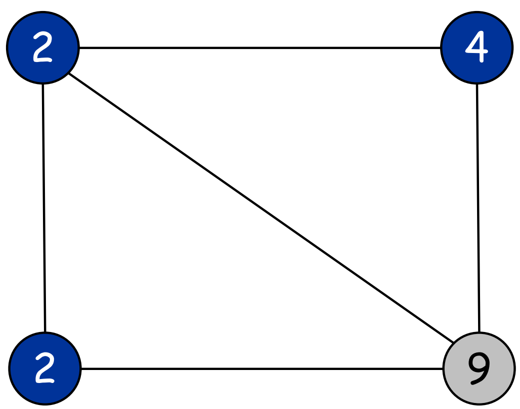
Thm. [Feige 1998] Unless P ≠ NP, no poly-time algorithm can do better than $1 - 1/e$.

Remark: Extended to submodular coverage

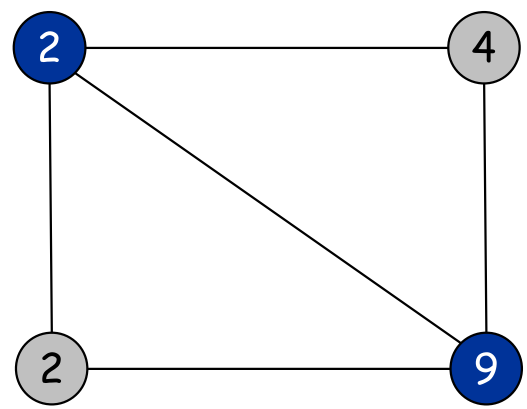
6. Weighted Vertex Cover

Weighted Vertex Cover

Weighted vertex cover. Given a graph $G = (V, E; c)$ with vertex costs, find a vertex cover of minimum cost.



cost = 8



cost = 11

Frugal bidding

Bidding: Each edge e offers a bid (payment) $p(e) \geq 0$ for coverage

Frugalness. $p(\delta(v)) \leq c(v)$ for each vertex v.

Claim. For any vertex cover S and any frugal bidding p, $c(S) \geq p(E)$.

Pf. $p(E) \leq \sum_{v \in S} p(\delta(v)) \leq \sum_{v \in S} c(v) = c(S)$

Tight vertices: $p\big(\delta(v)\big) = c(v)$ (cost is covered by the collectable bids)

- Only tight vertices are willing to join the VC

Frugal bidding method

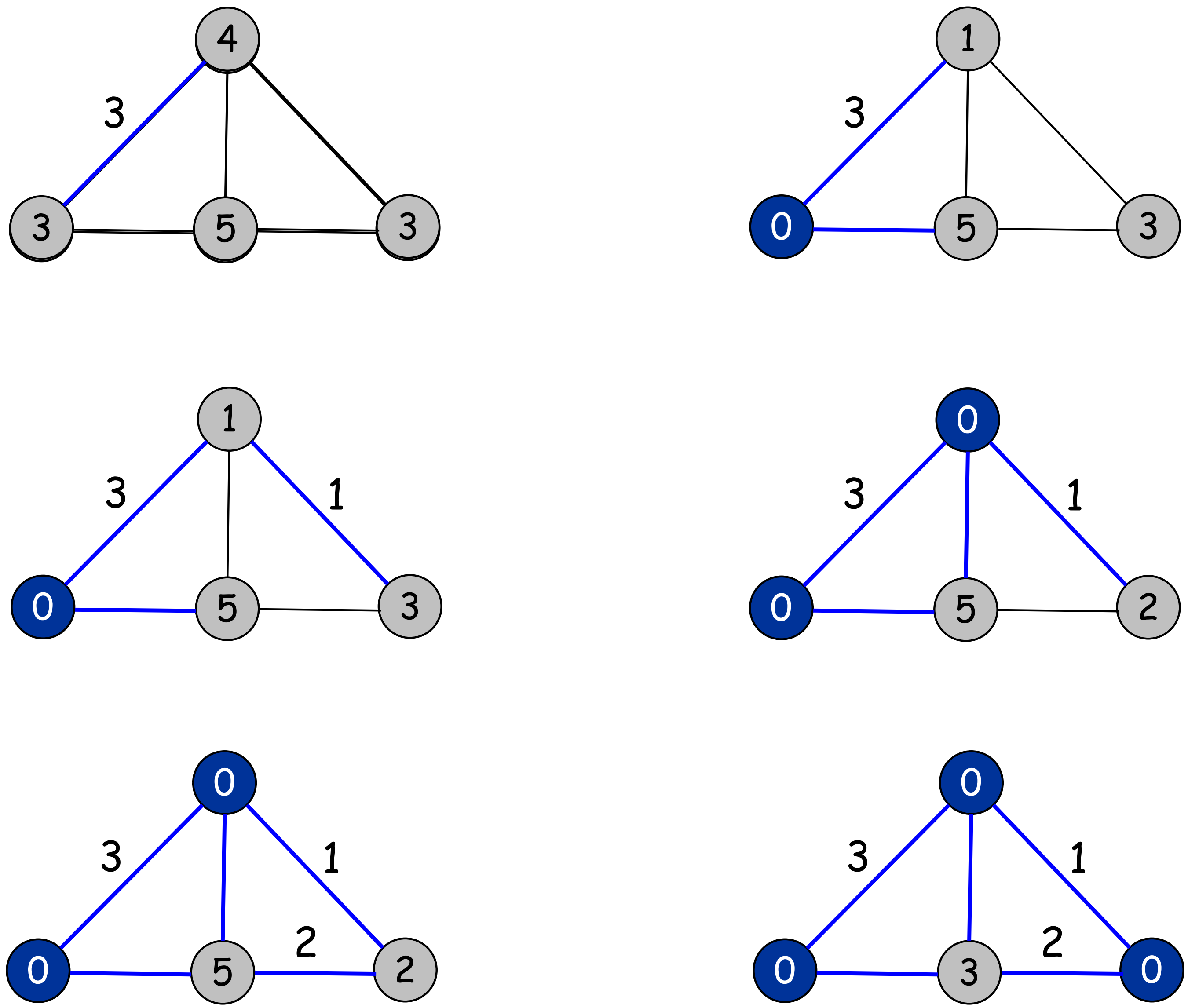
Set bids and find vertex cover simultaneously.

```
foreach e in E, p(e) ← 0
S ← ∅

while (∃ edge without tight endpoints)
   select such an edge e
   increase p(e) maximally without violating frugalness
   add tight endpoints of e to S

return S
```

Frugal bidding method: example



Frugal bidding method: analysis

Thm. Frugal bidding method is a 2-approximation.

Intuition: each edge's bid is competed by its 2 endpoints.

Pf. Let S^* be optimal vertex cover. We show $c(S) \le 2c(S^*)$.

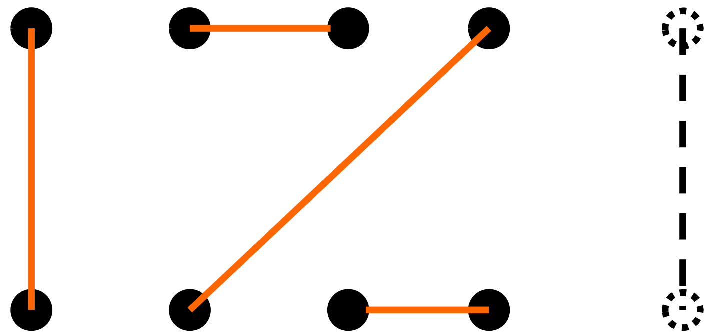
$$c(S) = \sum_{v \in S} c(v) = \sum_{v \in S} p(\delta(v))$$ ← all nodes in S are tight

$$\le \sum_{v \in V} p(\delta(v))$$

$$= 2p(E)$$ each edge counted twice

$$\le 2c(S^*)$$ frugalness lemma

(Unweighted) Vertex Cover: restriction



Compute a maximal matching M.

Return all matched vertices as a vertex cover.

$opt \geq |M|$

So, $2|M| \leq 2opt$, and we have a 2-approximation algorithm!

Approximation hardness of Minimum Vertex Cover

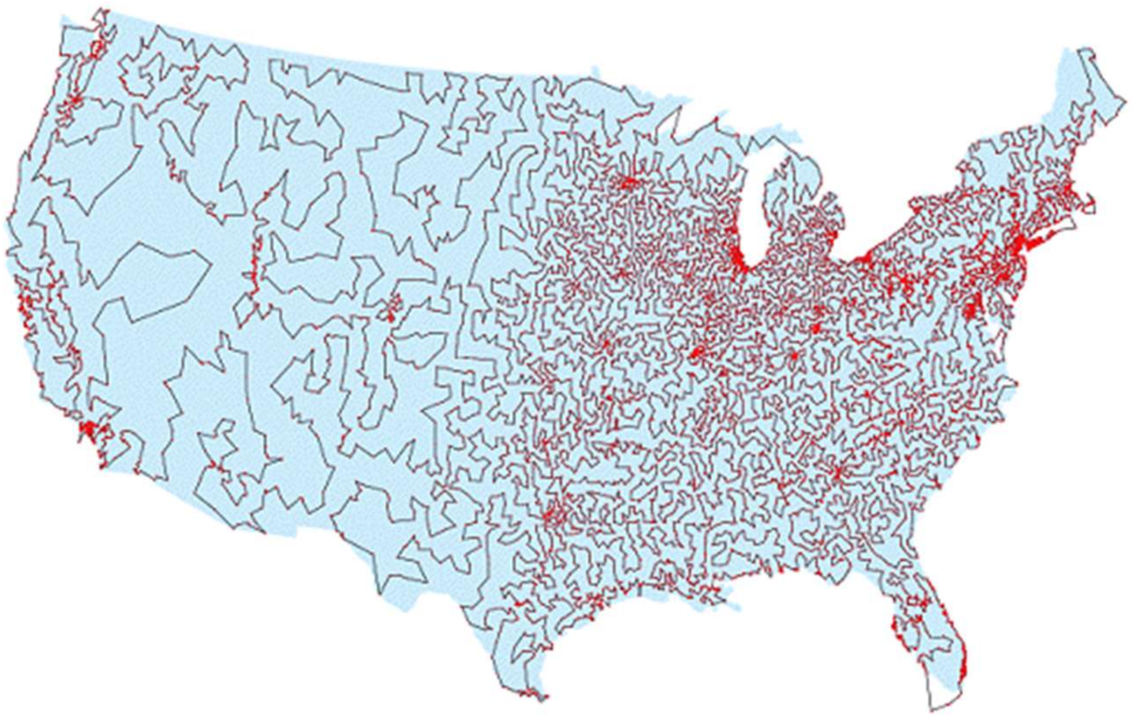
- NP-complete to approximate within a factor of 1.36
- No $(2-\varepsilon)$-approximation if the Unique Game Conjecture is true.
- Maximal IS: hard to approximate within $n^{1-\varepsilon}$.

7. Metric TSP

Metric Traveling Salesman Problem (TSP)

Input: n cities $c_1, \ldots, c_n$. with metric mutual distances

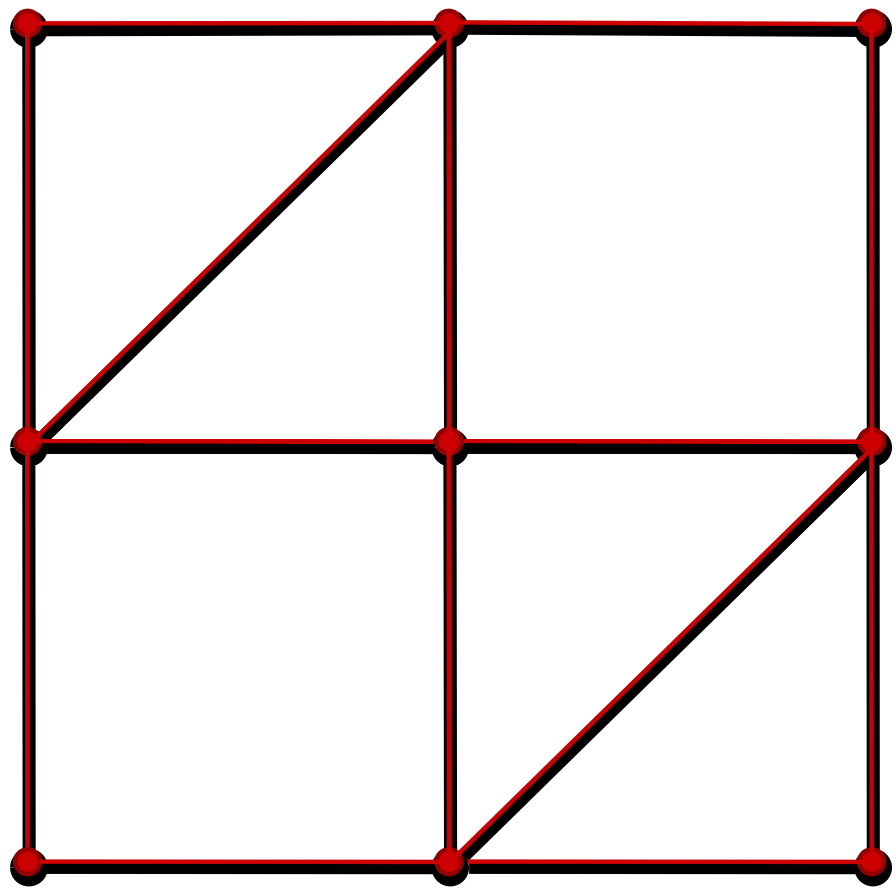
TSP: Find a Hamiltonian tour of shortest total length on the n cities



n = 13,509

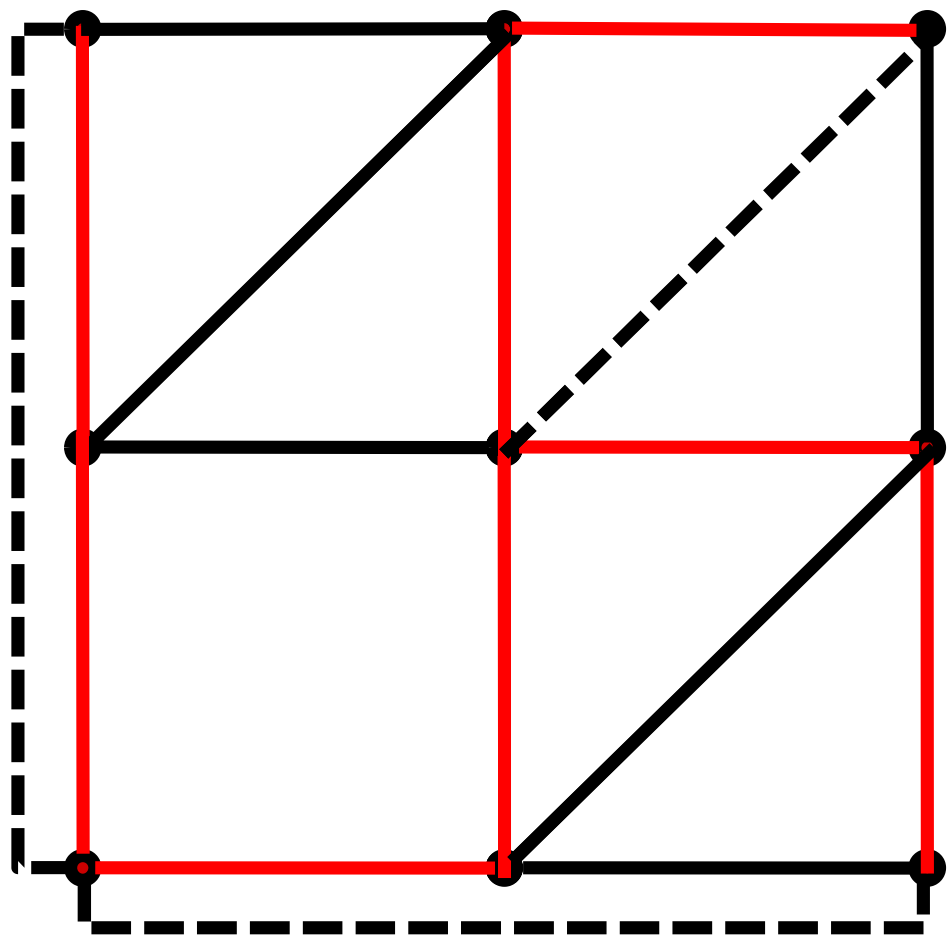
Relaxing Hamiltonian tour to spanning cycle

Relaxation: allow vertex repetitions in the tour



Extracting Hamiltonian tour from spanning cycle

Traversing a spanning cycle but skipping previously-visited vertices yields a Hamiltonian tour of no greater length



Constructing a short spanning cycle

Christofides–Serdyukov algorithm [Christofides, 1976]:

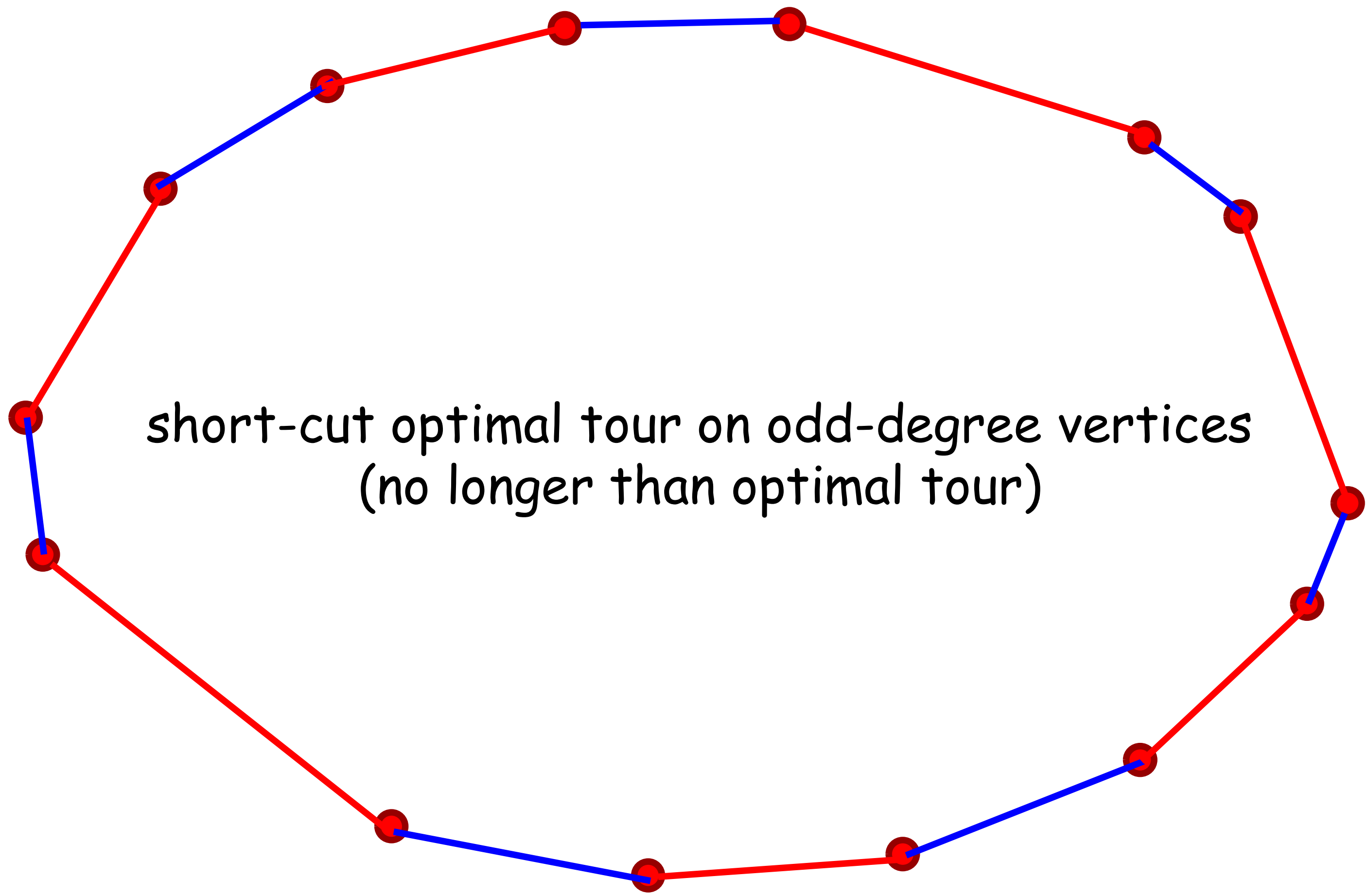
an MST + a shortest perfect matching on its odd-degree vertices

= a spanning cycle

Lemma. mst $\leq$ opt, matching $\leq$ opt/2

Theorem : Christofides –Serdyukov algorithm is a 3/2-approximation.

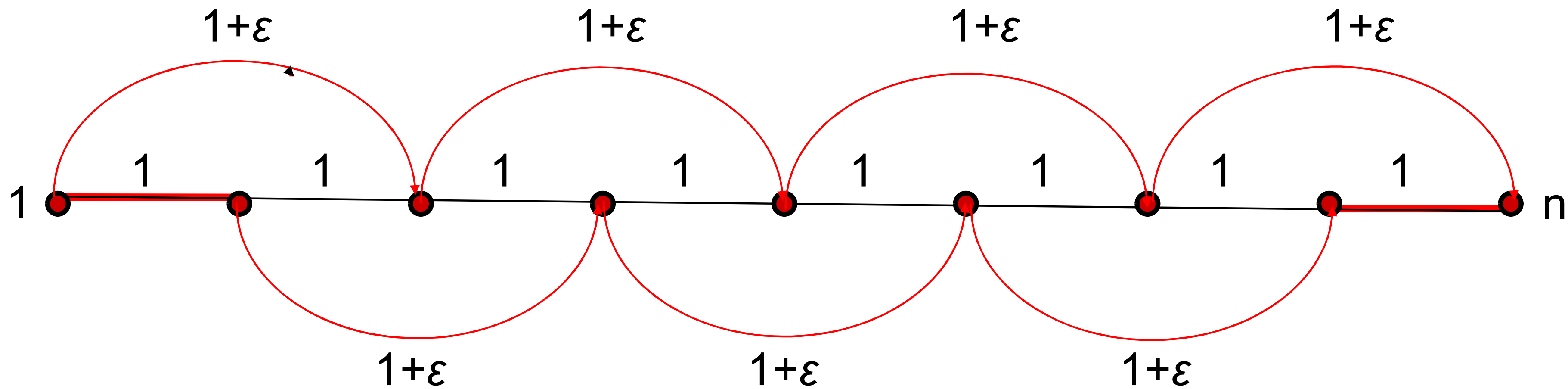
Bound on optimal matching



matching M_1 + matching M_2 = short-cut optimal tour

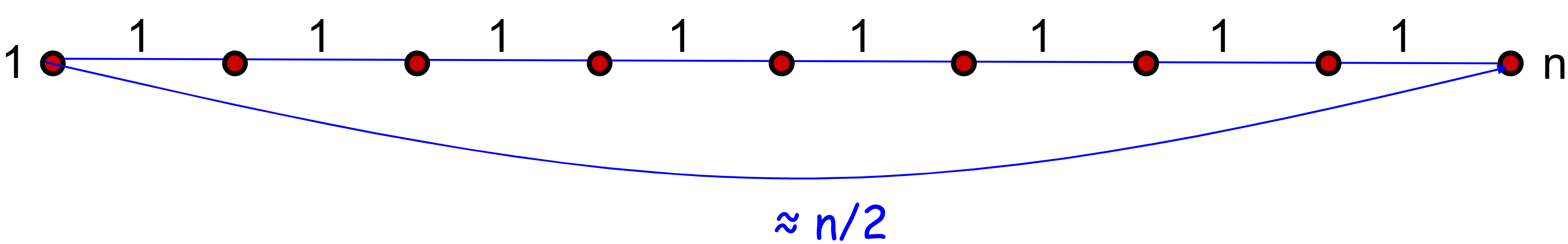
optimal matching $\leq \min(M_1, M_2) \leq$ opt/2

A tight instance



all other edges have distances given by the shortest paths

red tour ≈ n



output tour ≈ 1.5n

Can we do better?

- No better approx. algorithm is known.
- Assuming P $\neq$ NP, no polynomial-time algorithm can do better than 220/219 = 1.004566… [Papadimitriou & Vempala, 2006].
- For Euclidean and related metrics, there exists a PTAS [Arora, 1998][Mitchell, 1999].

8. Knapsack

Knapsack Problem

Input. n items with specified sizes and profits s_i and p_i; and a knapsack capacity B with $\max_i s_i \leq B < \sum_i s_i$

Def. The size and profit of a subset I of items:

$$s(I) = \sum_{i \in I} s_i \ , \qquad p(I) = \sum_{i \in I} p_i$$

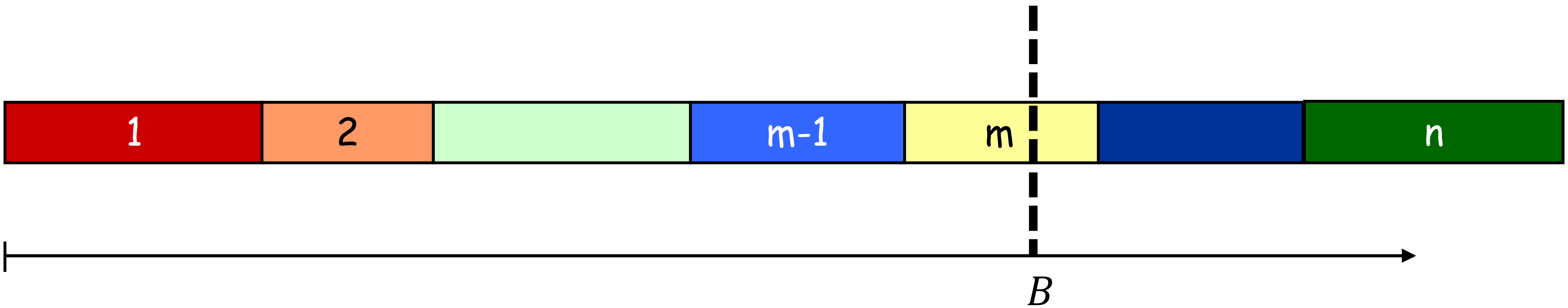
Def. For a subset I of items is feasible of $s(I) \leq B$

Knapsack problem: find a feasible subset with maximum profit.

Def. The return of an item i is $r_i = p_i / s_i$

Fractional Knapsack Problem

Fractional relaxation: allow a fraction of an item



Fractional greedy algorithm [Dantzig 1957]:

Sort the items by return in decreasing order: 1, 2, …, n.

Compute the first m s.t. $[1{:}m]$ is not feasible.

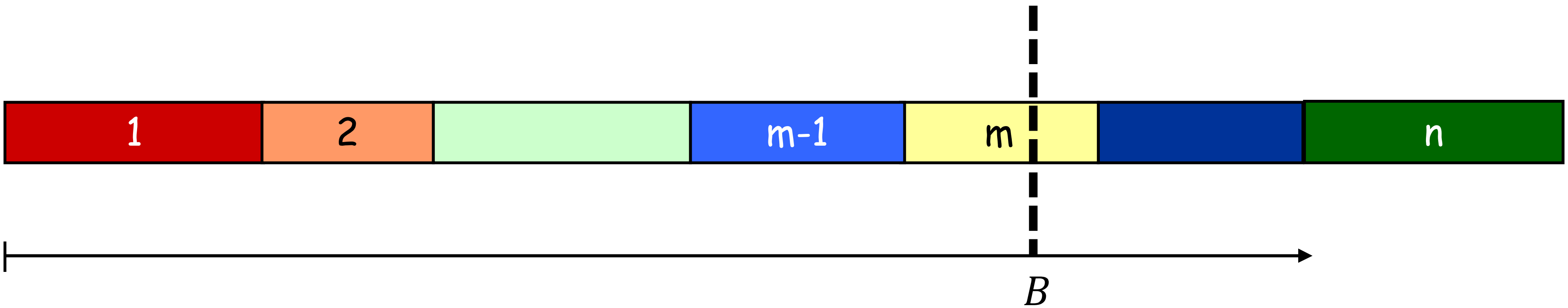
Add $[1{:}m-1]$ and a fraction $\{B - s([1{:}m-1])\}/s_m$ of item m

Fractional optimum:

$$opt^* = p([1{:}m-1]) + \frac{B - s([1{:}m-1])}{s_m} p_m < p([1{:}m-1]) + p_m$$

Rounding the fractional greedy solution

Rounding: pick the more profitable I between $[1{:}m-1]$ and m



Thm. $p(I) \geq opt/2$.

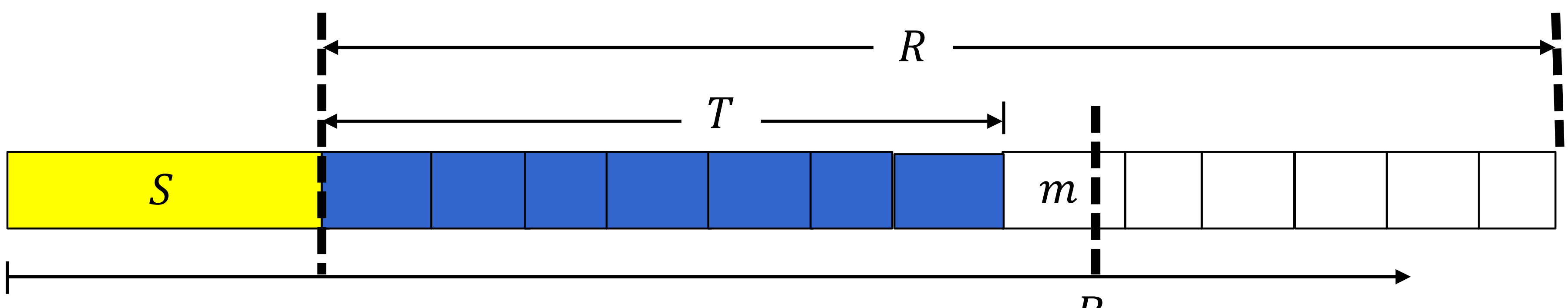
Pf. $opt \leq opt^* < p([1{:}m-1]) + p_m \leq 2p(I)$

A PTAS with partial enumeration

[Sahni 1975]

k: a fixed positive integer constant

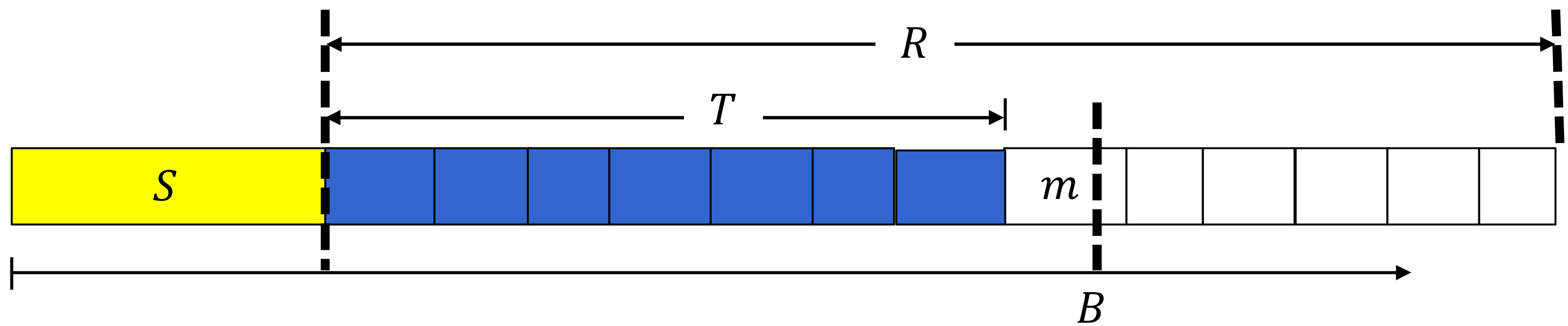
$I \leftarrow$ the best feasible subset of less than k items; //enumeration
for each feasible subset S of k items //enumeration
    $R \leftarrow \left\{ j \notin S : p_j \leq \min_{i \in S} p_i \right\}$ //pruning
    compute the greedy extension $T \subseteq R$ of S //greedy extension
    if $p(S \cup T) > p(I)$ then $I \leftarrow S \cup T$
return I.



Analysis of the PTAS

Thm. The running time is $O(n^{k+1})$, and $p(I) \geq opt/(1+1/k)$.

Pf. Assume I is not optimal. Let O be an optimal solution. Then $|O| > k$.
S: the set of k most profitable items in O



$$p(O \setminus S) < p(T) + p_m \leq p(T) + p(S)/k$$

$$opt = p(O) \leq p(T) + (1 + 1/k)p(S) \leq (1 + 1/k)p(S \cup T) \leq (1 + 1/k)p(I)$$

Recap: Dynamic programming for Knapsack

Assumption: the profits of all items are integers

Recap: solvable exactly by a 2-dim. DP in $O(n^2 P)$ time & space, where $P := \max_i p_i$

- States: min. size required to attain a profit q by the first k items.
- DP table indexed by (k, q): n rows and at most nP columns
- Each entry can be computed in constant time (look up two entries).

Pseudo-polynomial time & space

Optimal solution is preserved by scaling of the profits.

Rounding & Scaling

Idea: Round (up or down) and then scale down the profits.
Compute the optimal solution in this modified instance

- Suppose $P \geq 1000n$. Then $opt \geq P \geq 1000n$.
- Round up each profit to nearest multiple of 100 : $100\lceil p_i/100 \rceil$.
  - individual rounding error < 100.
  - total rounding error $< 100n < (1/10)opt$.
- Compute the optimal solution w.r.t the up-rounded profits:
  - scale down the up-rounded profits by 100 times: $p_i^* \coloneqq \lceil p_i/100 \rceil$
  - apply DP using this down-scaled profits
  - the running time is 100 times faster.

FPTAS for KNAPSACK

[Ibarra-Kim 1975]

$\varepsilon > 0$: a time-accuracy trade-off parameter

Let $K := \varepsilon P/n$; and define p^* by $p_i^* := \lfloor p_i/K \rfloor$ for $1 \leq i \leq n$.
Apply DP with p^* and to find the most profitable set I.
Return I.

Thm. $p(I) \geq (1-\varepsilon)opt$ and the running time is $O(n^3/\varepsilon)$.

Analysis of the FPTAS

Let O be an optimal set.

For each i, $p_i \le Kp_i^* = K\lceil p_i/K \rceil < p_i + K$.

$$opt = p(O) \le Kp^*(O) \le Kp^*(I) \le p(I) + nK = p(I) + \varepsilon P \le p(I) + \varepsilon opt$$

So, $p(I) \ge (1 - \varepsilon)opt$.

The DP with p^* has n rows and at most $n\lceil P/K \rceil = O(n^2/\varepsilon)$ columns.
So, the total time (and space) complexity is $O(n^3/\varepsilon)$.

Further speedup with reduced DP columns

The number of DP columns nP can be replaced any $C \geq opt$ and the DP runs $O(nC)$ time.

Choice of smaller C: Run the fractional greedy & rounding to output I_1, and let

$$C \coloneqq \min\{p([1:n]), 2p(I_1)\}$$

If $C \leq 2n/\varepsilon$, then DP returns an optimal solution in $O(n^2/\varepsilon)$ time.

So, assume $C > 2n/\varepsilon$.

Further speedup with reduced DP columns

$K := \varepsilon p(I_1)/n$
Define p^* by $p_i^* := \lfloor p_i/K \rfloor$ for $1 \leq i \leq n$ and $C^* = \lfloor C/K \rfloor$
Apply DP with p^* and C^* to find the most profitable set I_2.
Output the more profitable one I between I_1 and I_2.

$$C^* = \lfloor C/K \rfloor \leq 2p(I_1)/K \leq 2n/\varepsilon$$

So, the total time (and space) complexity is $O(n^2/\varepsilon)$.

Thm. $p(I) \geq opt/(1+\varepsilon\ )$
Pf. For each i, $p_i - K < Kp_i^* = K\lfloor p_i/K \rfloor \leq p_i$.

$$p(O) < Kp^*(O) + |O|K \leq Kp^*(I_2) + nK \leq p(I_2) + \varepsilon p(I_1) \leq (1+\varepsilon\ )p(I)$$

Quick Summary on FPTAS

1. Modify the instance by rounding/scaling the numbers.

2. Use DP to compute an optimal solution S in the modified instance.

3. Output S as the approximate solution.

Other examples:

- Load balancing with fixed number of machines,
- Other variants of Knapsack
- Delay constrained shortest path