Approximation Algorithms (Ch 11)

- **Approximation Algorithms, Definitions**
- **Approximation Algorithms for Minimization Problems**

  Load Balancing Problem, Center Selection Problem,

  Set Cover Problem, Vertex Cover Problem
- **Approximation Algorithms for Maximization Problems**

  Knapsack Problem, Maximum Pairwise Edge-disjoint Paths Problem
- **Polynomial Time Approximation Scheme (PTAS)**
- **Fully Polynomial Time Approximation Scheme (FPTAS)**
- **Inapproximability**

The lecture notes/slides are adapted from those associated with the text book by J. Kleinberg and E. Tardos.

Approximation Algorithms, Definitions

- **Optimization problem: for any problem instance x**
  - **there is a set $S(x)$ of feasible solutions,**
  - **each solution in $S(x)$ is associated with a value,**
  - **an optimal solution has the optimal (minimum/maximum) associated value.**
- **Every optimization problem has a corresponding decision problem:**

  Given a parameter k, whether the optimization problem has a solution with the associated value at most k (for minimization problem) or at least k (for maximization problem).

- **To solve an NP-hard problem, we need to sacrifice one of following:**
  - **Find an optimal solution of the problem.**
  - **Find a solution for every instance of the problem.**
  - **Find a solution of the problem in polynomial time.**
- **Strategies for coping NP-hard problems**
  - **Design algorithms to find approximate solutions.**
  - **Design algorithms for special cases of the problems.**
  - **Design algorithms which may take exponential time.**

- X, **optimization problem**

  A, **algorithm for** X

  opt, **the value associated with an optimal solution (**$\mathrm{opt} > 0$**)**

  S_A, **the value associated with a solution computed by** A **for** X

- **Algorithm** A **is an** **α-approximation algorithm** **for a minimization problem** X **if** A **finds a solution for any instance** x **of** X **s.t.** $S_A \leq \alpha \mathrm{opt}$ **in** $\mathrm{Poly}(|x|)$ **time.**

- **Algorithm** A **is an** **α-approximation algorithm** **for a maximization problem** X **if** A **finds a solution for any instance** x **of** X **s.t.** $S_A \geq \alpha \mathrm{opt}$ **in** $\mathrm{Poly}(|x|)$ **time.**

- **For minimization problems,** $\alpha \geq 1$**, and for maximization problems,** $0 < \alpha \leq 1$**.**

- **A minimization problem X admits a polynomial-time approximation scheme (PTAS) if for any $\epsilon > 0$, X admits a $(1+\epsilon)$-approximation algorithm.**
- **A maximization problem X admits a polynomial-time approximation scheme (PTAS) if for any $\epsilon > 0$ X admits a $(1-\epsilon)$-approximation algorithm.**
- **PTAS runs in $\mathrm{Poly}(|x|)$ time but may not run in $\mathrm{Poly}(1/\epsilon)$ time.**
- **A minimization problem X admits a fully polynomial-time approximation scheme (FPTAS) if for any $\epsilon > 0$ X admits a $(1+\epsilon)$-approximation algorithm which runs in $\mathrm{Poly}(1/\epsilon)$ time.**
- **A maximization problem X admits a fully polynomial-time approximation scheme (FPTAS) if for any $\epsilon > 0$ X admits a $(1-\epsilon)$-approximation algorithm which runs in $\mathrm{Poly}(1/\epsilon)$ time.**
- **FPTAS runs in $\mathrm{Poly}(|x|)$ time and $\mathrm{Poly}(1/\epsilon)$ time.**

Approximation Algorithms for Load Balancing

Load balancing problem

- **Given a set S of n jobs and m identical machines,**
  - **each job $j \in S$ must be processed continuously on one machine,**
  - **each job $j \in S$ can be processed on any machines in t_j time,**
  - **a machine can process at most one job at a time,**

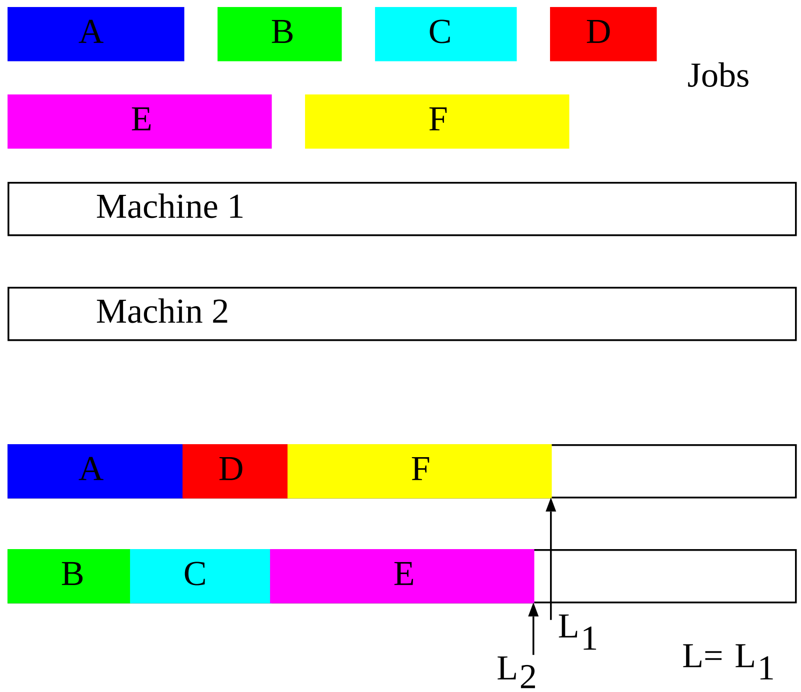
  find a schedule to assign the jobs of S to m machines.

- **Given a schedule, let $J(i)$ be the set of jobs assigned to machine i.**

  The load of machine i is $L_i = \sum_{j \in J(i)} t_j$.

  The length or makespan of the schedule is $L = \max_{1 \leq i \leq m} L_i$.

- **Optimization goal, minimizing the makespan L (NP-complete, Subset-Sum$\leq_P$Problem).**


A
B
C
D
Jobs
E
F
Machine 1
Machin 2
A
D
F
B
C
E
L1
L2
L= L1

A greedy algorithm for load balancing [Graham 1966]

List-Scheduling($m, n, t_1, t_2, .., t_n$)

Input: m **machines and** n **jobs with processing time.**

Output: A schedule to assign the jobs to machines.

for $i = 1$ **to** m **do**

  $L_i = 0$**;** /* **load on machine** i */

  $J(i) = \emptyset$**;** /* **jobs assigned to machine** i */

end for

for $j = 1$ **to** n **do**

  $i = \arg\min_{1 \leq k \leq m} \min L_k$**;** /* **machine** i **has the smallest load** */

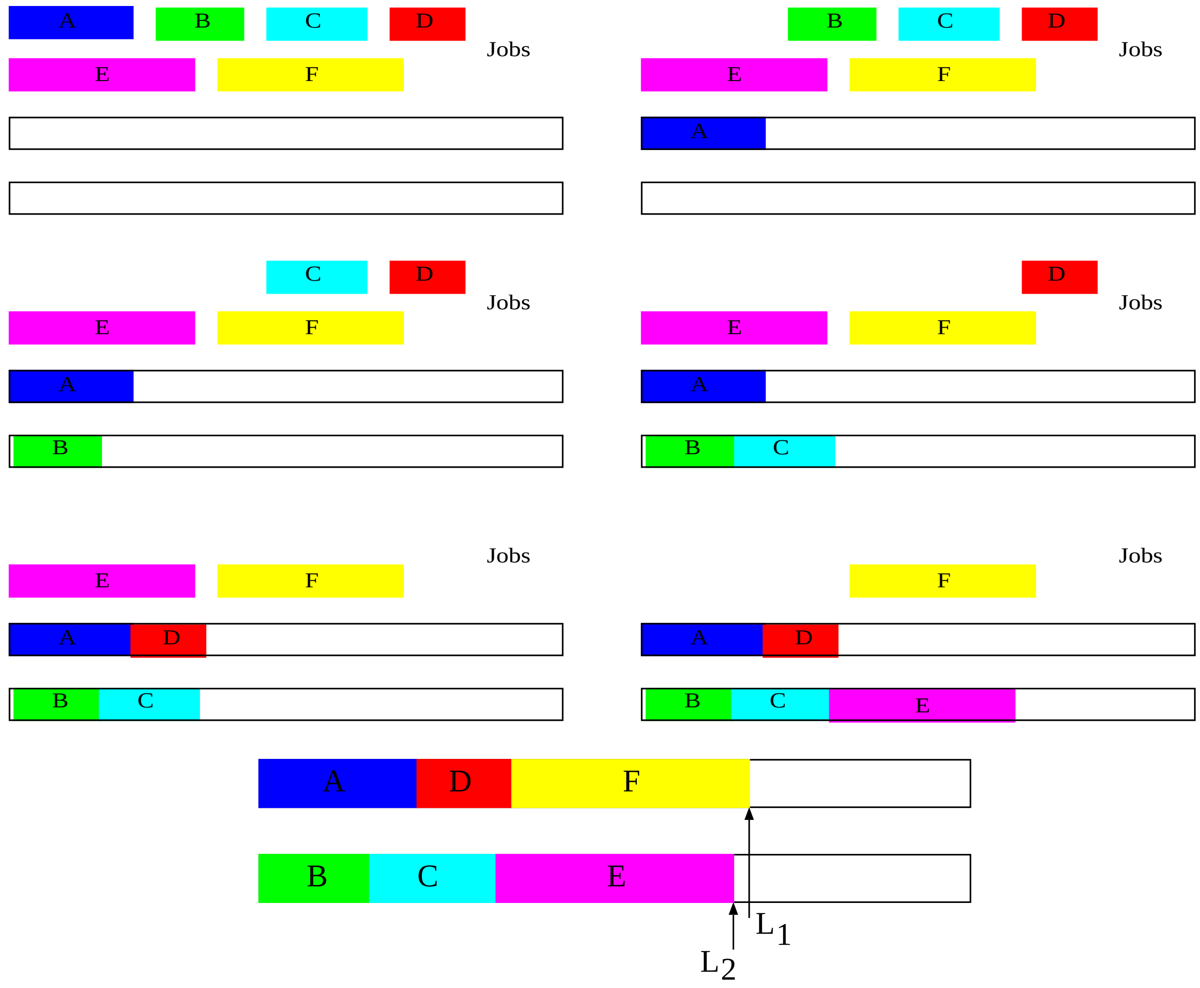
  $J(i) = J(i) \cup \{j\}$**;** /* **assign job** j **to machine** i */

  $L_i = L_i + t_j$**;** /* **update load of machine** i */

end for

Output $J(i)$ **for** $1 \leq i \leq m$

Implemention, $O(n \log n)$ time using a priority queue.


A
B
C
D
Jobs
E
F
L1
L2

Theorem. [Graham 1966] List-Scheduling algorithm is a 2-approximation algorithm

Proof. Let $L = \max_{1 \le i \le m} L_i$ be the makespan of the schedule by the algorithm and opt be the optimal makespan. We show that $L \le 2\mathrm{opt}$.

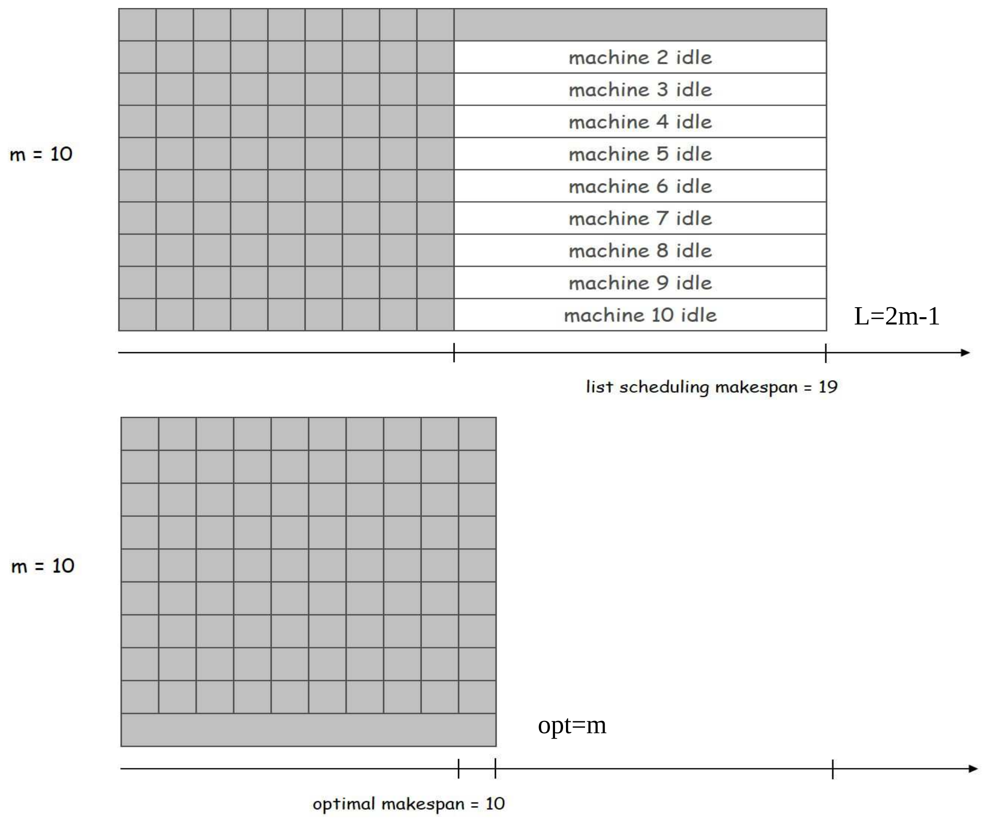
Observe that $\mathrm{opt} \ge \max_{1 \le j \le n} t_j$ and $\mathrm{opt} \ge \frac{1}{m} \sum_{1 \le j \le n} t_j$.

Assume $L_i = L$ and l is the last job assigned to machine i. Then $L_i - t_l \le L_k$ for all $1 \le k \le m$. Sum inequalities over all k,

$$L_i - t_l \le \frac{1}{m} \sum_{1 \le k \le m} L_k = \frac{1}{m} \sum_{1 \le j \le n} t_j \le \mathrm{opt}.$$

Now $L = L_i = (L_i - t_l) + t_l \le 2\mathrm{opt}$. $\square$

- **The approximation ratio of List-Scheduling algorithm is close to tight.**
- **Example, m machines, $n = m(m-1)$ jobs of processing time 1, and 1 job of processing time m.**



An improved algorithm for load balancing, longest processing time (LPT): sort jobs in descending order of processing time and then run List-Scheduling algorithm.

LPT-List-Scheduling($m, n, t_1, t_2, .., t_n$**)**

Input: m **machines and** n **jobs with processing time.**

Output: A schedule to assign the jobs to machines.

    Sort jobs so that $t_1 \geq t_2 \geq .. \geq t_n$**;**

    for $i = 1$ **to** m **do**

        $L_i = 0$**;** /* **load on machine** i */

        $J(i) = \emptyset$**;** /* **jobs assigned to machine** i */

    end for

    for $j = 1$ **to** n **do**

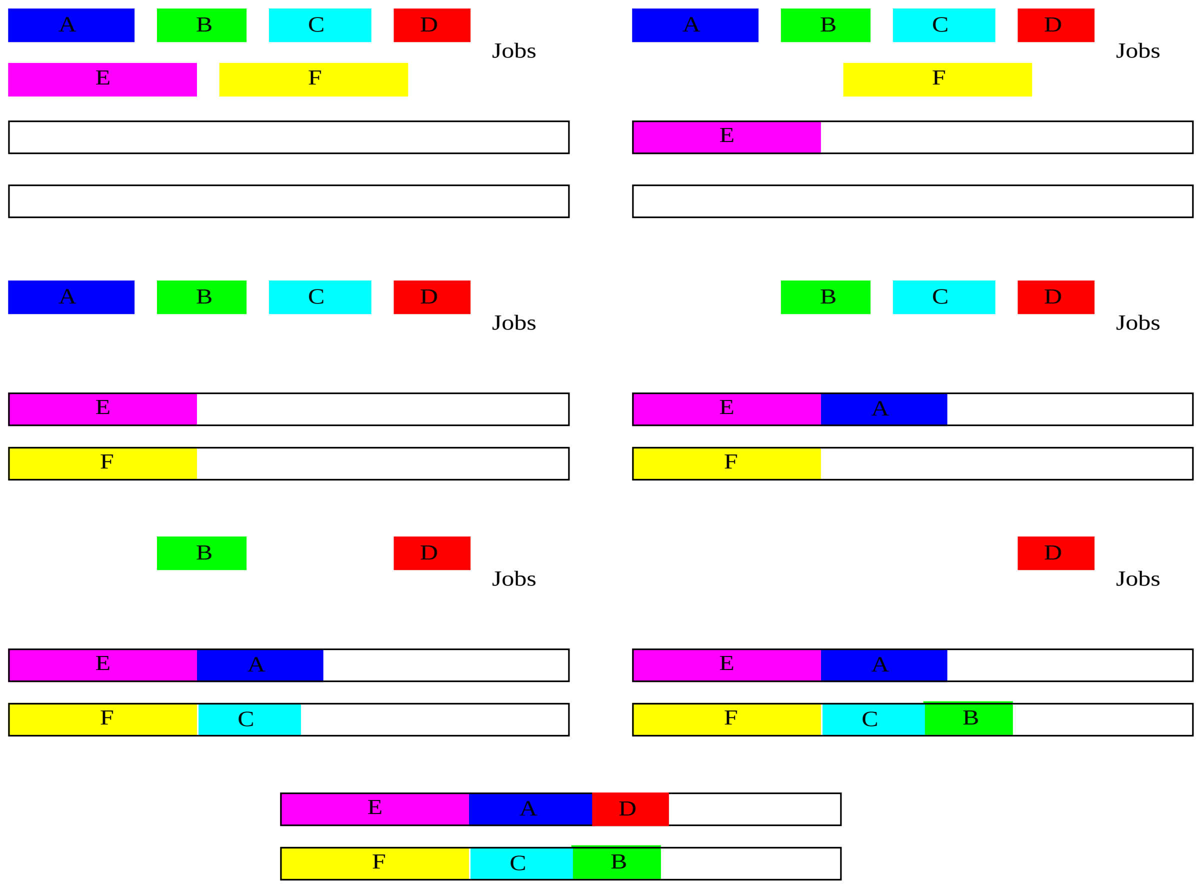
        $i = \arg\min_{1 \leq k \leq m} \min L_k$**;** /* **machine** i **has the smallest load** */

        $J(i) = J(i) \cup \{j\}$**;** /* **assign job** j **to machine** i */

        $L_i = L_i + t_j$**;** /* **update load of machine** i */

    end for

    Output $J(i)$ **for** $1 \leq i \leq m$


A B C D
Jobs
E F
A B C D
Jobs
F
E
A B C D
Jobs
E
F
B C D
Jobs
E A
F
B D
Jobs
E A
F C
D
Jobs
E A
F C B
E A D
F C B

LPT-List-Scheduling algorithm is a (3/2)-approximation algorithm.

Proof. Let $L = \max_{1 \le i \le m} L_i$ be the makespan of the schedule by the algorithm and opt be the optimal makespan. We show that $L \le (3/2)\mathrm{opt}$.

If $n \le m$, then $L = \mathrm{opt}$, otherwise $t_{m+1} \le (1/2)\mathrm{opt}$.

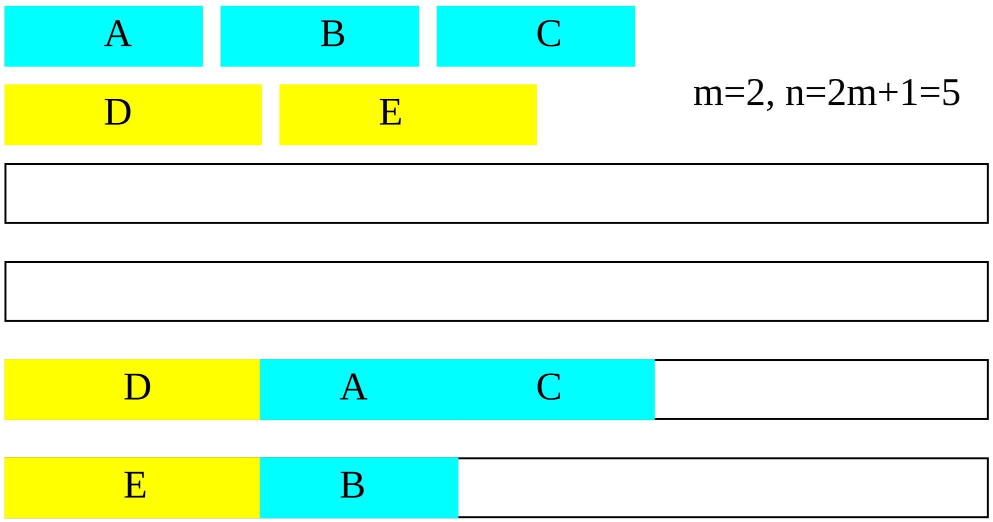
Recall, $\mathrm{opt} \ge \frac{1}{m} \sum_{1 \le j \le n} t_j$.

Assume $L_i = L$ and l is the last job assigned to machine i. Then $L_i - t_l \le L_k$ for all $1 \le k \le m$. Sum inequalities over all k,

$$L_i - t_l \le \frac{1}{m} \sum_{1 \le k \le m} L_k = \frac{1}{m} \sum_{1 \le j \le n} t_j \le \mathrm{opt}.$$

Now $L = L_i = (L_i - t_l) + t_l \le (3/2)\mathrm{opt}$ because $t_l \le t_{m+1} \le (1/2)\mathrm{opt}$. $\square$

- **The approximation ratio (3/2) of LPT-List-Scheduling algorithm is not tight.**
- **LPT-List-Scheduling algorithm is a (4/3)-approximation algorithm [Graham 1969].**
- **The approximation ratio (4/3) is close to tight.**
- **Example,** m **machines,** $n = 2m + 1$ **jobs, 3 jobs for processing time** m**, 2 jobs for each of processing time** $m + 1, .., 2m - 1$**. Then** $L \leq ((4m - 1)/(3m))\mathrm{opt}$**.**



Solution by LPT-List-Scheduling

Optimal solution

Approximation Algorithm for Center Selection

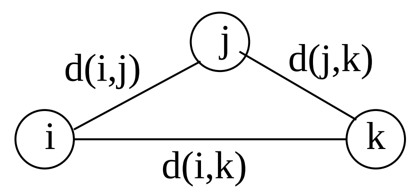
- **Center selection: Given a set $V = \{s_1, .., s_n\}$ of sites, find k centers (sites) s.t. the maximum distance from any site to the nearest center is minimized.**

- **Let $d(s_i, s_j)$ be the distance from s_i to s_j satisfy:**

  $d(s_i, s_i) = 0$ **for each** $s_i \in V$**.**

  $d(s_i, s_j) = d(s_j, s_i)$ **for** $s_i, s_j \in V$**.**

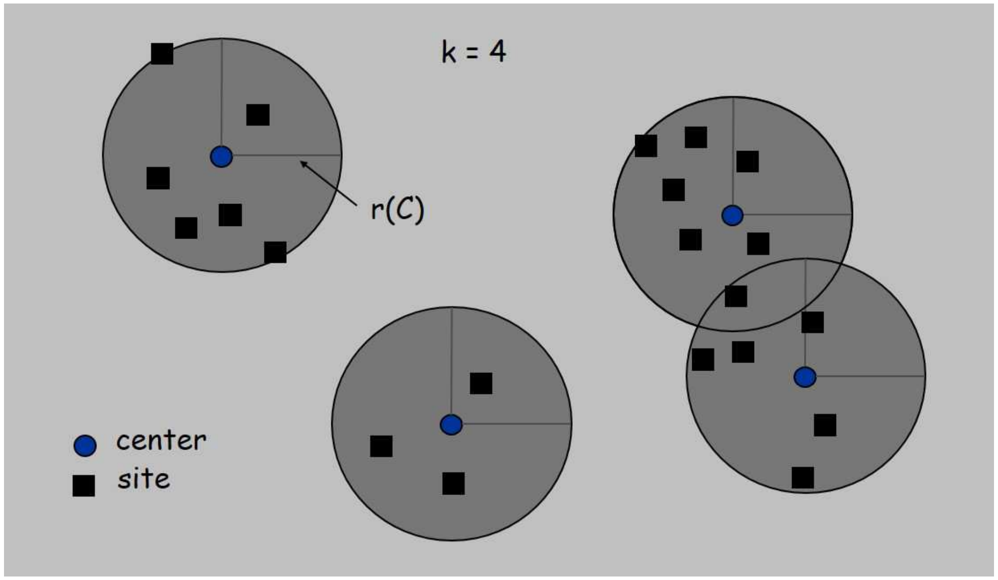
  $d(s_i, s_k) \leq d(s_i, s_j) + d(s_j, s_k)$ **for** $s_i, s_j, s_k \in V$**.**



- **Model input as a weighted complete graph G, $V(G) = \{s_1, .., s_n\}$, each edge $\{s_i, s_j\}$ has a distance $d(s_i, s_j) \geq 0$.**

  Given $C \subseteq V(G)$, $d(s_i, C) = \min_{s_j \in C} d(s_i, s_j)$ and $r(C) = \max_{s_i \in V(G)} d(s_i, C)$ is the radius of C.

- **Center selection problem: given G and integer k, find a subset C with $|C| = k$ s.t. the radius of C is minimized.**



A greedy algorithm for center selection

Greedy-Center-Selection(k, G)

Input: A weighted complete graph G and integer k.

Output: A subet $C \subseteq V(G)$ with $|C| = k$ and $r(C)$ minimized.

$C = \emptyset$;

repeat k **times;**

  select v_i **with maximum** $d(v_i, C)$;

  $C = C \cup \{v_i\}$;

end repeat

Return C

For any $v_i, v_j \in C$, $d(v_i, v_j) \geq r(C)$.

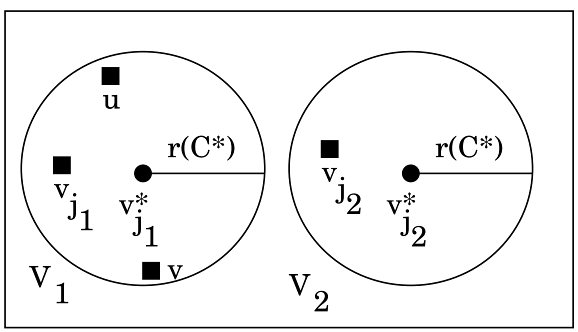
Theorem. Greedy-Center-Selection algorithm is a 2-approximation algorithm.

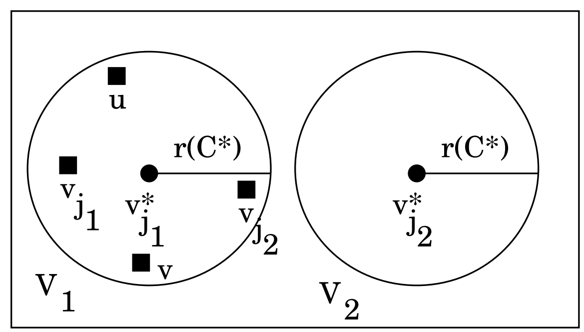
Proof. Let $C^* = \{v^*_{j_1}, ..., v^*_{j_k}\}$ be an optimal solution and $C = \{v_{j_1}, .., v_{j_k}\}$ be the solution found by the algorithm. We prove $r(C) \leq 2r(C^*)$.

Let $V_1, ..., V_k$ be a parition of $V(G)$ s.t. $v^*_{j_i} \in V_i$ and each $u \in V(G)$ is in V_i if $i = \arg\min_{v^*_{j_i} \in C^*} d(u, v^*_{j_i})$. For u, v in any V_i, $d(u, v) \leq 2r(C^*)$.

If $v_{j_i} \in V_i$ for $1 \leq i \leq k$, then for each $u \in V(G)$, there is a vertex $v_{j_i} \in C$ s.t. $d(u, v_{j_i}) \leq 2r(C^*)$, implying $r(C) \leq \max_{\substack{u \in V_i \\ 1 \leq i \leq k}} d(u, v_{j_i}) \leq 2r(C^*)$.

Assume $v_{j_1}, v_{j_2} \in C$ are in a same V_i. Since $d(v_{j_1}, v_{j_2}) \leq 2r(C^*)$, $r(C) \leq d(v_{j_1}, v_{j_2}) \leq 2r(C^*)$. $\square$





Approximation Algorithms for Set Cover

- **Set cover problem: given a set of n elements $U = \{u_1, ..., u_n\}$ and subsets $S_1, .., S_m$ ($S_j \subseteq U$, $1 \leq j \leq m$, $\cup_{1 \leq i \leq m} S_i = U$), find an $I \subseteq \{1, ..., m\}$ s.t. $\cup_{j \in I} S_j = U$ and $|I|$ is minimized.**

- **Greedy-Set-Cover**

  $X = U$ **/* X, set of uncovered elements.**

  $I = \emptyset$ **/* I, set of indices of the selected subsets in solution.**

  while $X \neq \emptyset$ do

  $j = \arg\max_{1 \leq j \leq m} |X \cap S_j|$**.** **/* $|X \cap S_j|$ is maximum.**

  $I = I \cup \{j\}$ **and** $X = X \setminus S_j$**.**

  end while

  Retrun I

- **Theorem: Greedy-Set-Cover is a $(1 + \ln n)$-approximation algorithm for the set-cover problem.**

  Proof. Let I^* be an optimal solution and $|I^*| = k$. For $n = 1$, $|I| = |I^*|$ and the theorem holds. For $n > 1$, we prove that $|I| \leq \lceil k \ln n \rceil$, implying $|I| \leq (1 + \ln n)k$. For $1 \leq i \leq |I|$, let $x_i = |X|$ before the ith iteration and $n_i = |X \cap S_j|$ in the ith iteration of the while loop. Then

  $$x_1 = n, x_{i+1} = x_i - n_i, \text{ and } n_i \geq \frac{x_i}{k}.$$

  The last inequality holds because the elements of X are covered by k subsets in optimal solution, there exists an $r \in I^*$ s.t. $|X \cap S_r| \geq \frac{|X|}{k} = \frac{x_i}{k}$; and the algorithm selects S_j with $|X \cap S_j|$ maximized ($|X \cap S_j| \geq |X \cap S_r|$).

By induction and $n_i \geq \frac{x_i}{k}$, we show that for $1 \leq i \leq |I|$, $x_i \leq n(1-1/k)^{i-1}$.
For $i = 1$, $x_1 = n = n(1-1/k)^{i-1}$. Assume $x_{i-1} \leq n(1-1/k)^{i-2}$ for $i-1 \geq 1$. Then
$x_i = x_{i-1} - n_{i-1} \leq x_{i-1} - \frac{x_{i-1}}{k} = x_{i-1}(1-1/k) \leq n(1-1/k)^{i-1}$.
From this, we claim $|I| \leq \lceil k \ln n \rceil$. Assume for contradiction that $|I| > \lceil k \ln n \rceil$.
Let $|I| = t$. Then $t \geq 1 + k \ln n$ and $x_t \leq n(1-1/k)^{k \ln n} < ne^{-\ln n} = 1$.
Since x_t is an integer, $x_t = 0$ and the algorithm terminates before iteration t. $\square$

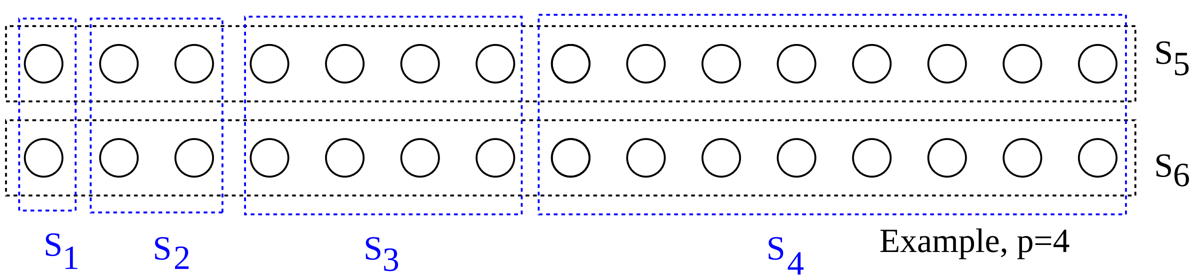
- **Greedy-Set-Cover algorithm is tight**

  Let $q = \sum_{j=0}^{p-1} 2^j = 2^p - 1$**,** $U = \{a_i, b_i | 1 \leq i \leq q\}$ **and subsets** $S_1, .., S_p, S_{p+1}, S_{p+2}$**, where for** $1 \leq j \leq p$**,**

  $S_j = \{a_i, b_i | 2^{j-1} \leq i \leq 2^j - 1\}$**,**

  $S_{p+1} = \{a_i | 1 \leq i \leq q\}$ **and** $S_{p+2} = \{b_i | 1 \leq i \leq q\}$**.**

  Then before each iteration $1 \leq j \leq p$**,** $|X \cap S_{p-j+1}| = 2 \times 2^{p-j+1}$ **and** $|X \cap S_{p+1}| = |X \cap S_{p+2}| = 2 \times 2^{p-j+1} - 1$**. So the solution by the algorithm is** $I = \{1, .., p\}$ **and the optimal solution is** $I^* = \{p+1, p+2\}$**. Let** $|U| = n$**. Then** $n = 2q = 2^{p+2} - 2$ **and** $|I|/|I*| = p/2 \geq (1/2)(\log n - 2)$**.**



- **Weighted set cover problem: given a set of n elements $U = \{u_1, ..., u_n\}$ and subsets $S_1, .., S_m$ ($S_j \subseteq U$, $1 \leq j \leq m$), and a weight $w(S_j) \geq 0$ for each S_j, find an $I \subseteq \{1, ..., m\}$ such that $\cup_{j \in I} S_j = U$ and $\sum_{j \in I} w(S_j)$ is minimized.**

- **Greedy-Weighted-Set-Cover**

  $X = U$ **/* X, set of uncovered elements.**

  $I = \emptyset$ **/* I, set of indices of the selected subsets in solution.**

  while $X \neq \emptyset$ do

  $j = \arg\min_{1 \leq j \leq m} \frac{w(S_j)}{|X \cap S_j|}$. **/* S_j has minimum cost per uncovered element.**

  $I = I \cup \{j\}$ **and** $X = X \setminus S_j$.

  end while

  Retrun I

- **Theorem: Greedy-Weighted-Set-Cover is an H_n-approximation algorithm for the weighted set cover problem, where $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the nth harmonic number.**

  Proof. For input $U = \{u_1, .., u_n\}, S_1, .., S_m$, let $d = \max_{1 \le j \le m} |S_j|$. We prove Greedy-Weighted-Set-Cover is an H_d-approximation algorithm.

  Let I be the solution of the algorithm and rename the subsets of the solution as $S'_1, .., S'_{|I|}$ s.t S'_i is selected at the ith iteration. For each element $u_l \in X \cap S'_i$, we charge a cost $c(u_l) = \frac{w(S'_i)}{|X \cap S'_i|}$. Each element of U is charged exactly once and $\sum_{i \in I} w(S'_i) = \sum_{1 \le l \le n} c(u_l)$.

Let S be any of $S_1, .., S_m$ and $r = |S|$. Rename elements of S as $u'_1, .., u'_r$ in the order they are covered by the algorithm. For each element $u'_l \in S$, u'_l is covered by some S'_i, $i \in I$, and $\{u'_l, u'_{l+1}, .., u'_r\} \subseteq X$. So at the ith iteration, $\frac{w(S'_i)}{|X \cap S'_i|} \leq \frac{w(S)}{|X \cap S|}$, implying $c(u'_l) \leq \frac{w(S)}{|X \cap S|} \leq \frac{w(S)}{r-l+1}$. Then

$$\sum_{1 \leq l \leq r} c(u'_l) \leq w(S) \sum_{1 \leq l \leq r} \frac{1}{r-l+1} = w(S) H_r.$$

Let I^* be an optimal solution.

$$\begin{aligned} \sum_{i \in I} w(S'_i) &= \sum_{1 \leq l \leq n} c(u_l) = \sum_{i* \in I*} \sum_{u'_l \in S_{i*}} c(u'_l) \\ &\leq \sum_{i* \in I*} w(S_{i*}) H_{|S_{i*}|} \leq H_d \sum_{i* \in I*} w(S_{i*}). \end{aligned}$$

□

Approximation Algorithms for Vertex Cover

- **Vertex Cover of graph G is a special case of Set Cover**

  Let U the set of edges of G.

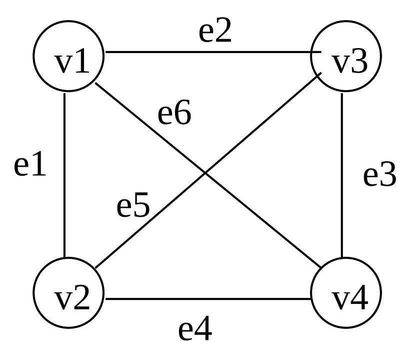
  For each vertex v_i, let S_i be the set of edges incident to v_i and $w_i = 1$.

  A minimum set cover is a minimum vertex cover of G.

- **Example, for G in the figure**

  $U = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, $S_1 = \{e_1, e_2, e_6\}$, $S_2 = \{e_1, e_4, e_5\}$, $S_3 = \{e_2, e_3, e_5\}$, $S_4 = \{e_3, e_4, e_6\}$ **and** $w_1 = w_2 = w_3 = w_4 = 1$.

  $\{S_1, S_2, S_4\}$ **is minimum set cover and** $\{v_1, v_2, v_4\}$ **is a minimum vertex cover**



Approx-VC(G)

Input: Undirected graph G.

Output: A vertex cover S of G

$U = E(G)$; $S = \emptyset$;

while $U \neq \emptyset$ **do**

select an arbitrary edge $e = \{u, v\} \in U$;

$S = S \cup \{u, v\}$; /* $\{u, v\}$ **is viewed as subset of vertices**

remove every edge incident to u **or** v **from** U

end of while

return S

Algorithm Approx-VC is a 2-approximation algorithm

Proof. Obviously, the algorithm finds a vertex cover S of G in $O(|E(G)|)$ time. Let S^* be a minimum vertex cover of G, we show $|S| \leq 2|S^*|$.

Let E' be the set of edges selected in the while loop of the algorithm. Then S^* contains at least one end vertex of every $e \in E'$, implying $|S^*| \geq |E'|$. Since $|S| = 2|E'|$, $|S| \leq 2|S^*|$. $\square$

Approximation Algorithms for Weighted Vertex Cover

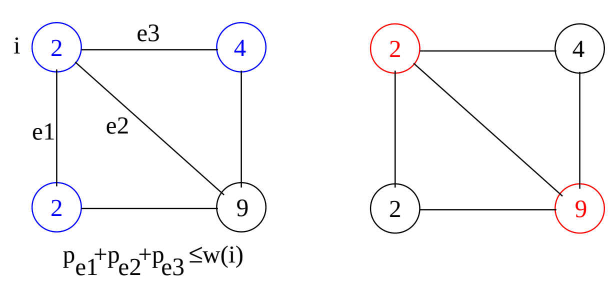
- **Weighted vertex cover: given a graph G with each node u assigned a weight $w(u) \geq 0$, find a vertex cover S of G s.t. $w(S) = \sum_{u \in S} w(u)$ is minimized.**
- **Pricing mehod: Each edge must be covered by some node i; edge e pays price p_e to use i.**

  Fairness: Edges incident to a same node i should pay at most $w(i)$ in total, that is, for each node i, $\sum_{e=\{i,j\}} p_e \leq w(i)$.
- **Claim: For any vertex cover S and any fair prices p_e, $\sum_{e \in E(G)} p_e \leq w(S)$.**

Proof.

$$\sum_{e \in E(G)} p_e \leq \sum_{i \in S} \sum_{e=\{i,j\}} p_e \leq \sum_{i \in S} w(i) = w(S).$$

$\square$



Price method for weighted vertex cover [Bar-Yehuda and Even 1981]

Weighted-Vertex-Cover-Price-Method(G)

Input: A node weighted graph G.

Output: A min-weight vertex cover S of G.

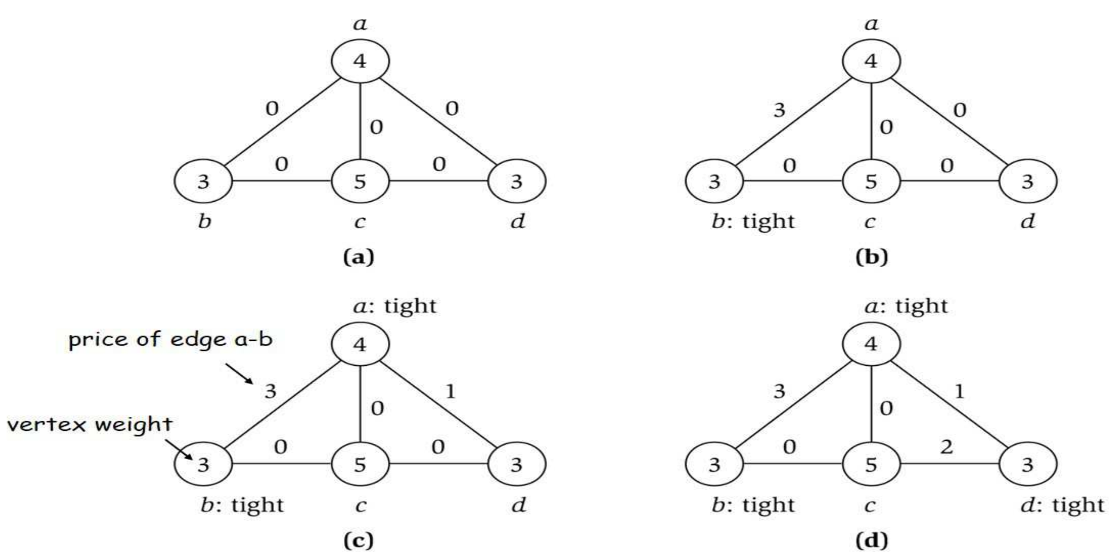
for each edge e of G do $p_e = 0$;

while $\exists e = \{i, j\}$ s.t. neighter i nor j is tight do/* i is tight if $\sum_{e=\{i,j\}} p_e = w(i)$ */

select such an edge $e = \{i, j\}$;

increase p_e as much as possible until i or j becomes tight;

Return the set of tight nodes as S



Theorem. [Bar-Yehuda and Even 1981] Pricing method is a 2-approximation algorithm for the weighted vertex cover problem.

Proof. Algorithm terminates since at least one node becomes tight in one iteration of the while loop.

Let S be the set of tight nodes. If some edge $\{i, j\}$ is not covered by S, then neighther i nor j is tight, contradicting with the termination of the algorithm. So S is a vertex cover.

Let S^* be an optimal vertex cover. We show $w(S) \leq 2w(S^*)$.

$$w(S) = \sum_{i \in S} w(i) = \sum_{i \in S} \sum_{e=\{i,j\}} p_e \leq \sum_{i \in V} p_e = 2 \sum_{e \in E} p_e \leq 2w(S^*).$$

$\square$

Linear programming (LP) method for weighted vertex cover

- **For each vertex $v \in V(G)$, let $x(v)$ be a binary variable that $x(v) = 1$ if v is included in a vertex cover, otherwise $x(v) = 0$. Edge $\{u, v\}$ is covered by a vertex cover if $x(u) + x(v) \geq 1$.**

- **0-1 integer program for finding a minimum weighted vertex cover of G**

minimize $$\sum_{v \in V(G)} x(v) \cdot w(v)$$

subject to $x(u) + x(v) \geq 1$ **for every** $\{u, v\} \in E(G)$

$x(v) \in \{0, 1\}$ **for every** $v \in V(G)$

- **Linear programming relaxation for weighted vertex cover**

minimize $$\sum_{v \in V(G)} x(v) \cdot w(v)$$

subject to $x(u) + x(v) \geq 1$ **for every** $\{u, v\} \in E(G)$

$0 \leq x(v) \leq 1$ **for every** $v \in V(G)$

LP method for weighted vertex cover

Weighted-Vertex-Cover-LP-method(G)

Input: A node weighted graph G.

Output: A min-weight vertex cover S of G.

$S = \emptyset$**;**

Solve the Linear programming relaxation to get $\{x(v)|v \in V(G)\}$;

for each $x(v)$ if $x(v) \geq \frac{1}{2}$ then $S = S \cup \{v\}$;

return S

Theorem. LP method is a 2-approximation algorithm for the weighted vertex cover problem.

Proof. The LP relaxation can be solved in polynomial time to get $\{x(v)|v \in V(G)\}$. For every edge $\{u, v\}$ of G, because $x(u) + x(v) \geq 1$ in the LP relaxation, $x(u) \geq \frac{1}{2}$ or $x(v) \geq \frac{1}{2}$, implying $u \in S$ or $v \in S$. Therefore, the algorithm finds is a vertex cover S of G.

Let S^* be an optimal vertex cover of G. We prove $w(S) \leq 2w(S^*)$. Let $w^* = \sum_{v \in V(G)} x(v) \cdot w(v)$ for the optimal solution in the LP relaxation. Because S^* is also a solution for the LP relaxation, $w^* \leq w(S^*)$. We show $w(S) \leq 2w^* \leq 2w(S^*)$.

$$\begin{aligned} w(S) &= \sum_{v \in S} w(v) = \sum_{x(v) \geq 1/2} w(v) \\ &\leq \sum_{v \in V(G)} 2x(v) \cdot w(v) = 2w^*. \end{aligned}$$

$\square$

Approximation algorithm for Knapsack problem

- **Knapsack problem: Given a set $I = \{1, ..., n\}$ of items, each item i has a positive integer value v_i and a positive integer weight w_i, and a knapsack with a positive integer capacity $W \geq w_i$ for $i \in I$, find a subset $S \subseteq I$ s.t. $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.**

W = 11

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

Greedy approximation algorithm for Knapsack problem

Greedy-Knapsack(n, W)

sort items s.t. if $i < j$ **then** $v_i/w_i \geq v_j/w_j$**;**

$S = \emptyset$**;** $i = 1$**;** $x = 0$**;**

while $i \leq n$ **and** $x < W$ **do**

if $x + w_i \leq W$ **then** $\{S = S \cup \{i\}$**;** $x = x + w_i$**;** $i = i + 1; \}$

else $x = x + w_i$

if $x \leq W$ **then return** S

else if $\sum_{j=1}^{i} v_j > v_{i+1}$ **then return** S **else return** $S = \{i + 1\}$

Theorem. Greedy-Knapsack is a $(\frac{1}{2})$-approximation algorithm for Knapsack problem.

Proof. The algorithm takes $O(n \log(T + W))$ time to find a solution for Knapsack problem, where $T = \sum_{i \in I} v_i$.

Let t^* be the value of an optimal solution and t be value of a solution by the algorithm. We show that $t \geq t^*/2$. If $S = I$ then $t = t^*$. Assume for some $i < n$, S is $\{1, .., i\}$ or $S = \{i + 1\}$. Then $t^* \leq (\sum_{j=1}^{i} v_j) + v_{i+1}$. Hence
$t = \max\{\sum_{j=1}^{i} v_i, v_{i+1}\} \geq t^*/2$. $\square$

Approximation algorithm for maximum pairwise edge-disjoint paths problem

- **Given k node pairs $(s_1, t_1), .., (s_k, t_k)$ in a digraph G, for each pair (s_i, t_i), a routing path for (s_i, t_i) is a path from s_i to t_i.**

  Two routing paths are edge-disjoint if they do not share a common edge.

  The problem asks to find a maximum number of edge-disjoint routing paths.

- **The max-flow technique can find a maximum number of edge-disjoint paths from $S = \{s_1, .., s_k\}$ to $T = \{t_1, .., t_k\}$, but a path may connect s_i to t_j for $i \neq j$, does not solve the problem.**

- **The problem is NP-complete (3-SAT$\leq_P$Problem)**

A greedy approximation algorithm

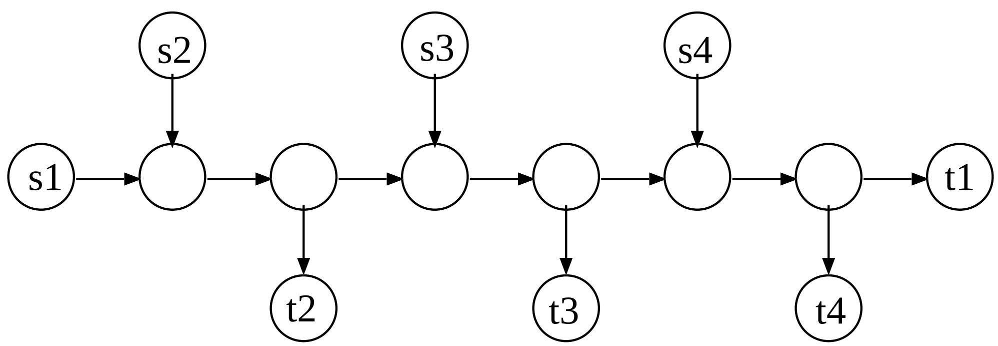
- **Idea**

  Compute a shortest path $s_i \to t_i$ for every pair (s_i, t_i).

  Include path $s_i \to t_i$ into solution in the increasing order of length (number of edges) until no path can be added.

- **Intuition**

  A shorter path uses less edges and has smaller chance to block other paths.



Greedy-Disjoint-Paths

Input: Node pairs $(s_1, t_1), .., (s_k, t_k)$ **in a digraph** G**.**

Output: Set $\mathcal{P}$ **of edge-disjoint paths, each path connecting a pair** (s_i, t_i)**.**

$\mathcal{P} = \emptyset$**;**

for $i = 1$ **to** k **do compute a shortest path** P_i **from** s_i **to** t_i**;**

for each path P_i **in increasing order of length do**

  if P_i **is edge-disjoint with every path of** $\mathcal{P}$ **then** $\mathcal{P} = \mathcal{P} \cup \{P_i\}$**;**

return $\mathcal{P}$

Theorem. Greedy-Disjoint-Paths is a $(\frac{1}{2\sqrt{m}+1})$-approximation algorithm for the maximum pairwise edge-disjoint paths problem in a digraph G of m edges.

Proof. Let $\mathcal{P}^*$ be an optimal solution. We prove $|\mathcal{P}| \geq \frac{|\mathcal{P}^*|}{2\sqrt{m}+1}$.

A path P is called a long path if P has at least $\sqrt{m}$ edges, otherwise a short path. Since G has m edges, $\mathcal{P}^*$ has at most $\sqrt{m}$ long paths. Let $\mathcal{P}_S$ and $\mathcal{P}^*_S$ be the sets of short paths in greedy solution and optimal solution, respectively.

Let P^*_i be a path in $\mathcal{P}^*_S$ but not in $\mathcal{P}$. Then there is a path $P_j \in \mathcal{P}$ that P_j blocks P^*_i. Since the algorithm selects paths in increasing order of their length, P_j is considered before P^*_i, implying P_j has at most $\sqrt{m}$ edges. Hence P_j can block at most $\sqrt{m}$ paths of $\mathcal{P}^*_S$. So, $\sqrt{m}|\mathcal{P}_S| \geq |\mathcal{P}^*_S|$ and

$$\begin{aligned}(2\sqrt{m}+1)|\mathcal{P}| &\geq \sqrt{m}|\mathcal{P}_S| + |\mathcal{P}| + \sqrt{m} \\ &\geq |\mathcal{P}^*_S| + |\mathcal{P}| + \sqrt{m} \geq |\mathcal{P}^*|.\end{aligned}$$

$\square$

Polynomial Time Approximaion Scheme (PTAS)

- **A minimization problem X admits a polynomial-time approximation scheme (PTAS) if for any $\epsilon > 0$, X admits a $(1+\epsilon)$-approximation algorithm.**
- **A maximization problem X admits a polynomial-time approximation scheme (PTAS) if for any $\epsilon > 0$ X admits a $(1-\epsilon)$-approximation algorithm.**
- **PTAS runs in $\mathrm{Poly}(|x|)$ time but may not run in $\mathrm{Poly}(1/\epsilon)$ time.**

PTAS for Knapsack problem

- **Idea**

  Partition items into high value ones and low value ones.

  Enumerate every subset of high value items and size $O(\frac{1}{\epsilon})$, if the weight of the subset is at most W, use Greedy-Knapsack algorithm to add low value items to the subset find a solution.

  Select a solution of maximum value from the solutions.

  Running time polynomial in n but exponential in $O(\frac{1}{\epsilon})$.

- **Given $\epsilon > 0$, an item i is high value if $v_i \geq \epsilon t$ otherwise low value, where t is an estimation of optimal solution value t^*.**

  (make solution of value at least $(1-\epsilon)t^*$)

For $A \subseteq I$**, let** $t(A) = \sum_{i \in A} v_i$ **and** $w(A) = \sum_{i \in A} w_i$**.**

PTAS-Knapsack(n, W, ϵ**)**

S =**Greedy-Knapack(**n, W**);** /* $I = \{1, .., n\}$**;**

$I_\epsilon = \{i | 1 \leq i \leq n, v_i \leq \epsilon \cdot t(S)\}$**;** **/* set of low value items**

Assume $I_\epsilon = \{1, .., m\}$ **and** $1, .., m$ **are in decreasing order of** $\frac{v_i}{w_i}$**;**

$\mathcal{S} = \{S' | S' \subseteq I \setminus I_\epsilon, |S'| \leq \frac{2}{\epsilon}\}$**;** **/* subsets of high values items**

for each $S' \in \mathcal{S}$ **do**

  if $w(S') > W$ **then** $t(S') = 0$

  else

    S'_ϵ**=Greedy-Knapsack**$(m, W - w(S'))$**;** /* $I_\epsilon = \{1, .., m\}$

    $S' = S' \cup S'_\epsilon$**;**

Return $\hat{S} = \arg\max_{S' \in \mathcal{S}} t(S')$**;**

Theorem. PTAS-Knapsack is a $(1-\epsilon)$-approximation algorithm for Knapsack problem.

Proof. Let S^* be an optimal solution, $\hat{S}$ be the solution by PTAS-Knapsack and S be the solution by Greedy-Knapsack. We show $t(\hat{S}) \geq (1-\epsilon)t(S^*)$.

Let $S^*_h = \{i \in S^* | v_i > \epsilon \cdot t(S)\}$. Since $v_i > \epsilon \cdot t(S)$ for every $i \in S^*_h$ and $t(S) \geq \frac{t(S^*)}{2}$, if $|S^*_h| > \frac{2}{\epsilon}$ then

$$t(S^*_h) > (\frac{2}{\epsilon})\epsilon \cdot t(S) = 2t(S) \geq t(S^*),$$

contradiction as $t(S^*)$ is the optimal value. So, $|S^*_h| \leq \frac{2}{\epsilon}$ and $S^*_h \in \mathcal{S}$.

Let S^*_ϵ be an optimal solution and S'_ϵ be the solution by Greedy-Knapsack for the Knapsack problem $I_\epsilon = \{1, .., m\}$ and bag capacity $W - w(S^*_h)$. Then $t(S^*) = t(S^*_h) + t(S^*_\epsilon)$. If $S'_\epsilon = I_\epsilon$ then $t(S'_\epsilon) = t(S^*_\epsilon)$.

Assume $t(S'_\epsilon) = \max\{(\sum_{j=1}^{i} v_i), v_{i+1}\}$. Then

$$t(S^*_\epsilon) \leq t(S'_\epsilon) + v_{i+1} \leq t(S'_\epsilon) + \epsilon \cdot t(S) \leq t(S'_\epsilon) + \epsilon \cdot t(S^*),$$

implying $t(S^*_\epsilon) - \epsilon \cdot t(S^*) \leq t(S'_\epsilon)$. Therefore,

$$t(\hat{S}) \geq t(S^*_h) + t(S'_\epsilon) \geq t(S^*_h) + t(S^*_\epsilon) - \epsilon \cdot t(S^*) = (1 - \epsilon)t(S^*).$$

There are at most $n^{2/\epsilon}$ subsets in $\mathcal{S}$. The algorithm takes $O(n^{1+2/\epsilon} \log(nTW))$ time. $\square$

PTAS for load balancing

- **For job $j \in S$, t_j is the processing time of j on any machine.**

  For $k \geq 1$, job l is called short job if $t_l \leq \frac{1}{km} \sum_{j \in S} t_j$, otherwise long job.

Algorithm A_k for scheduling

Find an optimal schedule for long jobs by enumeration;

Let S_S be the set of short jobs, S_L be the set of long jobs;

Let $L(i) = \sum_{j \in S_L \text{ assigned to machine } i} t_j$

be the load of machine i for long jobs in optimal schedule;

while $S_S \neq \emptyset$ do

$a := \arg\min_{i=1}^{m} L(i)$;

Assign job $j \in S_S$ to machine a and $S_S := S_S \setminus \{j\}$;

$L(a) := L(a) + t_j$;

end while

- **For $l \in S_L$, $t_l > \frac{1}{km} \sum_{j \in S} t_j$ and $\sum_{l \in S_L} t_l \leq \sum_{j \in S} t_j$. Thus, $|S_L| < km$.**
- **It takes $O(m^{km})$ time to find an optimal schedule for long jobs. Step 2 takes $O(mn)$ time. Running time of Algorithm A_k is $O(m^{km} + mn)$.**
- **Let L be the maksapn of the algorithm and l be the last job to complete. Then**

$$L - t_l \leq \frac{1}{m} \sum_{j \neq l} t_j.$$

Notice that $\mathrm{opt} \geq \frac{1}{m} \sum_{j \in S} t_j$. If l is a short job, then $t_l \leq \frac{1}{km} \sum_{j \in S} t_j$ and

$$L \leq \frac{1}{km} \sum_{j \in S} t_j + \frac{1}{m} \sum_{j \neq l} t_j \leq (1 + \frac{1}{k})(\frac{1}{m}) \sum_{j \in S} t_j \leq (1 + \frac{1}{k})\mathrm{opt}.$$

If l is a long job, then $L = \mathrm{opt}$.

- **The family of Algorithms $\{A_k\}$ is a PTAS when m is a constant.**

PTAS for arbitrary m.

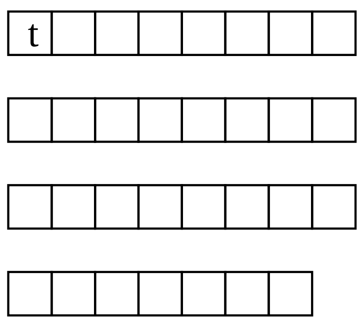
- **Idea: assign each machine at most k long jobs to reduce $O(m^{km})$ to $O(m^{k^{O(1)}})$.**

  Consider $t = \frac{1}{km} \sum_{j \in S} t_j$ as one unit time. Each $l \in S_L$ has $t_l > t$, $|S_L| < km$ and $\sum_{l \in S_L} t_l \leq \sum_{j \in S} t_j = kmt$.

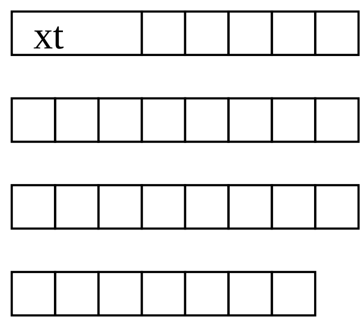
  Assign each machine i at most k long jobs with $L(i) \lessapprox kt$.

  If $\exists l^* \in S_L$ with $t_{l^*} \approx xt$ ($x > 1$ an integer) then $|S_L| < mk - x + 1$. If l^* is assigned to i, then i is assigned at most $k - x$ long jobs $l \neq l^*$.

- **Details: round t_l of long job to an integer in $\{k, k+1, .., k^2\}$, assign each machine at most k long jobs to reduce time from $O(m^{km})$ to $O(m^{k^{O(1)}})$.**



m=4, k=8, each long job has processing time t



If there is a job l* of processing time xt, 1<x≤k, job l* replaces x jobs of processing time t

- **For integer $k \geq 1$ and $T \geq \frac{1}{m}\sum_{j\in S} t_j$, job j is short job if $t_j \leq T/k$, otherwise long job. For each long job j, we modify (round) the processing time t_j to** $t'_j = \lfloor kt_j/(T/k) \rfloor = \left\lfloor t_j/(T/k^2) \right\rfloor$ $(t'_j(T/k^2) \leq t_j \leq t'_j(T/k^2) + T/k^2)$

- **Schedule A of makespan $\leq T$ for S_L with original processing time implies A has makespan $\leq T/(T/k^2)$ for S_L with modified processing time.**

- **Schedule A of makespan $\leq T/(T/k^2)$ for S_L with modified processing time implies A has makespan $\leq (1+1/k)T$ for S_L with original processing time.**

  Proof: Let S_i be the set of jobs assigned to any machine i in A. Since $t'_j(T/k^2) \leq t_j \leq t'_j(T/k^2) + T/k^2$ **for $j \in S_i$ and $|S_i| \leq k$,**

$$\sum_{j\in S_i} t_j \leq \sum_{j\in S_i} [t'_j(T/k^2) + (T/k^2)] \leq T + k(T/k^2) = (1+\frac{1}{k})T.$$

- **Use Step 2 of Algorithm A_k to schedule the set S_S of short jobs. Let l be the last job of S_S assigned to machine i. Since $T \geq \frac{1}{m}\sum_{j\in S}$, $T > \frac{1}{m}\sum_{j\in S, j\neq l} T$. From this, $L(i) < T$ before l is assigned to i and**

$$t_l + L(i) \leq t_l + \frac{1}{m}\sum_{j\in S, j\neq l} t_j < T/k + T = (1+\frac{1}{k})T.$$

- **Algorithm achieves approximation ratio $(1+\frac{1}{k})$.**

- **There is an algorithm which in $n^{O(k^2)}$ time finds a schedule of length$\leq T/(T/k^2)$ for S_L with the modified processing time. This gives a family of algorithms $\{B_k\}$ which is a PTAS for arbitrary m**

Fully Polynomial Time Approximation Scheme (FPTAS)

- **A minimization problem X admits a fully polynomial-time approximation scheme (FPTAS) if for any $\epsilon > 0$ X admits a $(1 + \epsilon)$-approximation algorithm which runs in $\mathrm{Poly}(1/\epsilon)$ time.**
- **A maximization problem X admits a fully polynomial-time approximation scheme (FPTAS) if for any $\epsilon > 0$ X admits a $(1 - \epsilon)$-approximation algorithm which runs in $\mathrm{Poly}(1/\epsilon)$ time.**
- **FPTAS runs in $\mathrm{Poly}(|x|)$ time and $\mathrm{Poly}(1/\epsilon)$ time.**

FPTAS for knapsack problem

- **Dynamic programming algorithm for the knapsack problem.**

  Dynamic-Knapsack(n, W)
  $A(1) := \{(0,0), (v_1, w_1)\}$;
  for $j := 2$ **to** n **do**
    $A(j) := A(j-1)$;
    for each $(t, w) \in A(j-1)$ **do**
      if $w + w_j \leq W$ **then** $A(j) := A(j-1) \cup \{(t + v_j, w + w_j)\}$;
    Remove dominated pairs from $A(j)$;
  Return a solution in $A(n)$ **with the maximum** t;

- **The algorithm solves the knapsack problem in $O(n \min\{T, W\})$ time,** $T = \sum_{i \in I} v_i$.

Reduce the running time of Dynamic-Knapsack

- **Intuition**

  Round all values up to lie in smaller range

  Run Dynamic-Knapsack on rounded instance

  Return optimal solution in rounded instance

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

original instance

Item	Value	Weight
1	2	1
2	7	2
3	19	5
4	23	6
5	29	7

W = 11

rounded instance

FPTAS-Knapsack(n, W, ϵ)
Let $M = \max_{i \in I} v_i$ **and** $\mu = \epsilon M/n$
for each $i \in I$ **do** $v'_i = \lfloor v_i/\mu \rfloor$**;** $/ *\ 0 \leq v'_i \leq v_i/\mu = (nv_i)/(\epsilon M) \leq n/\epsilon$
Run Dynamic-Knapsack on rounded values v'_i **for** $i \in I$ **to get a solution**

Theorem. FPTAS-Knapsack is an FPTAS for knapsack problem.

Proof. From $M = \max_{i \in I} v_i$, $\mu = \epsilon M/n$ and $v'_i = \lfloor v_i/\mu \rfloor$ for $i \in I$;

$$T' = \sum_{i \in I} v'_i = \sum_{i \in I} \left\lfloor \frac{v_i}{\epsilon M/n} \right\rfloor = O(n^2/\epsilon).$$

The algorithm to solve the modified instance in $O(n \min\{T', W\}) = O(n^3/\epsilon)$ time.

Let S^* be the optimal solution to original input and S be the optimal solution to the modified instance. From $v'_i = \lfloor v_i/\mu \rfloor$, $v'_i \leq v_i/\mu \leq v'_i + 1$ and $\mu v'_i \leq v_i \leq \mu(v'_i + 1)$, implying $\mu v'_i \geq v_i - \mu$.

From $\mathrm{opt} = \sum_{i \in S^*} v_i$, $M \leq \mathrm{opt}$ and $\mu = \epsilon M/n$,

$$\begin{aligned} \sum_{i \in S} v_i &\geq \mu \sum_{i \in S} v'_i \geq \mu \sum_{i \in S*} v'_i \geq \sum_{i \in S*} v_i - |S^*|\mu \geq \sum_{i \in S*} v_i - n\mu \\ &= \sum_{i \in S*} v_i - \epsilon M \geq \mathrm{opt} - \epsilon \mathrm{opt} = (1-\epsilon)\mathrm{opt}. \end{aligned}$$

□

Inapproximability

- **There is no ρ-approximation for the center selection problem for any $\rho < 2$ unless P=NP.**

Proof. We show that the following dominating set problem can be solved in poly-time by a ρ-approximation algorithm for the center selection problem.

  - A dominating set of a graph G is a subset $D \subseteq V(G)$ s.t. for any node u of G, either $u \in D$ or u is adjacent to a node $v \in D$.
  - Dominating set problem: given G and integer k, does G have a dominating set of size k?

For a dominating set instance G and k, we construct a center selection instance, a weighted complete graph H with $V(H) = V(G)$, $d(u, v) = 1$ if $\{u, v\} \in E(G)$, otherwise $d(u, v) = 2$.

Then G has dominating set size k iff H has k centers C^* with $r(C^*) = 1$. A ρ-approximation algorithm for $\rho < 2$ finds C^* in poly-time if C^* exists. □

- **TSP is not approximable unless P=NP.**

  Proof. We show that the Hamiltonian Cycle problem can be solved in poly-time if there is a $(1+\alpha)$-approximation algorithm for TSP.

  Let G be an instance of Hamiltonian Cycle. We construct a TSP instance H as

  - $V(G)$ is the set of cities,
  - $d(u,v) = 1$ if $\{u,v\} \in E(G)$, otherwise $d(u,v) = 2(1+\alpha)|V(G)|$.

  If G has a Hamiltonian cycle, then H has a tour of length $|V(G)|$, otherwise the minimum tour in H is at least $2(1+\alpha)|V(G)|$.

  A $(1+\alpha)$-approximation algorithm for TSP will give a cycle of length $\leq (1+\alpha)|V(G)|$ that is a Hamiltonian cycle.

  We can set α arbitrarily large. $\square$

- **For any $\epsilon > 0$, there is no $(1-\epsilon)\ln n$-approximation algorithm for the set cover problem unless P=NP.**
- **For any $\epsilon > 0$, there is no $(\frac{1}{m^{1/2-\epsilon}})$-approximation algorithm for the maximum pairwise edge-disjoint paths problem unless P=NP.**
- **Maximum Independent Set: Given a graph G, find a largest subset $S \subseteq V(G)$ s.t. for any $u, v \in S$, $\{u, v\} \notin E(G)$.**
- **Maximum Clique: Given a graph G, find a maximum clique (complete subgraph) of G.**
- **Max Independent Set and Max Clique are not approximable unless P=NP.**