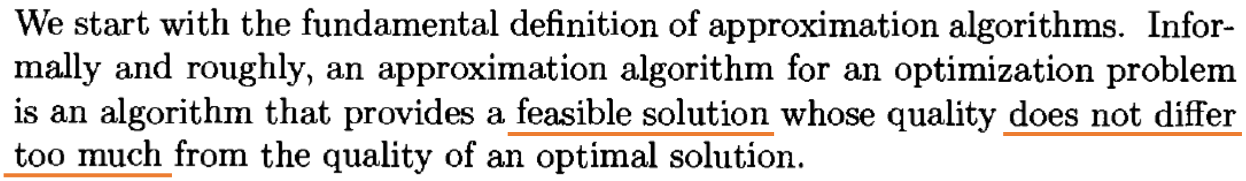
3-14 近似算法的基本概念

程龚

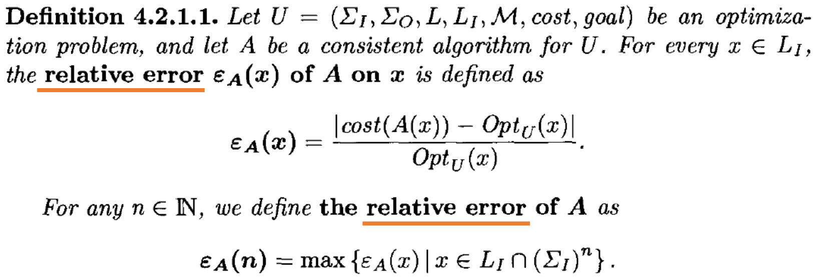
什么样的算法可以称作近似算法？

什么样的算法可以称作近似算法?



We start with the fundamental definition of approximation algorithms. Informally and roughly, an approximation algorithm for an optimization problem is an algorithm that provides a feasible solution whose quality does not differ too much from the quality of an optimal solution.

你能解释这些概念吗？



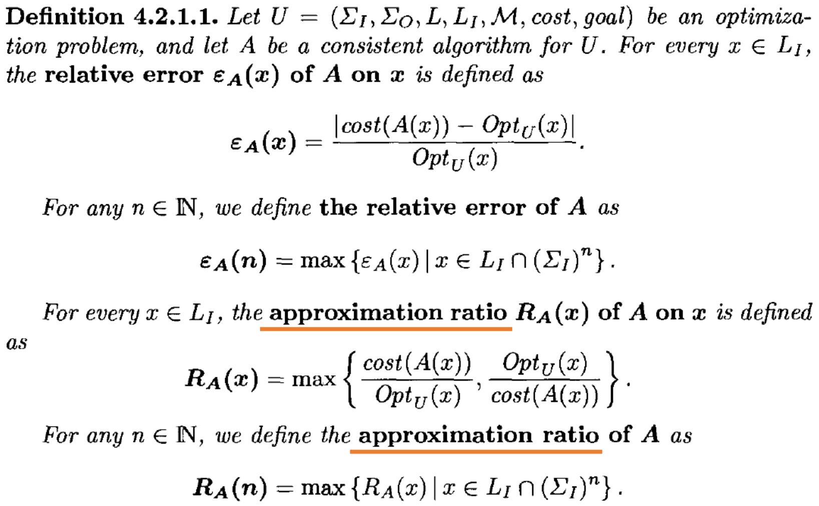
Definition 4.2.1.1. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *be an optimization problem, and let* A *be a consistent algorithm for* U. *For every* $x \in L_I$, *the* **relative error** $\boldsymbol{\varepsilon_A(x)}$ **of** $\boldsymbol{A}$ **on** $\boldsymbol{x}$ *is defined as*

$$\boldsymbol{\varepsilon_A(x)} = \frac{|cost(A(x)) - Opt_U(x)|}{Opt_U(x)}.$$

For any $n \in \mathbb{N}$, *we define* **the relative error of** $\boldsymbol{A}$ *as*

$$\boldsymbol{\varepsilon_A(n)} = \max\{\varepsilon_A(x) \,|\, x \in L_I \cap (\Sigma_I)^n\}.$$

你能解释这些概念吗？



Definition 4.2.1.1. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *be an optimization problem, and let* A *be a consistent algorithm for* U*. For every* $x \in L_I$*, the* **relative error** $\boldsymbol{\varepsilon_A(x)}$ **of** $\boldsymbol{A}$ **on** $\boldsymbol{x}$ *is defined as*

$$\boldsymbol{\varepsilon_A(x)} = \frac{|cost(A(x)) - Opt_U(x)|}{Opt_U(x)}.$$

For any $n \in \mathbb{N}$*, we define* **the relative error of** $\boldsymbol{A}$ *as*

$$\boldsymbol{\varepsilon_A(n)} = \max\{\varepsilon_A(x) \,|\, x \in L_I \cap (\Sigma_I)^n\}.$$

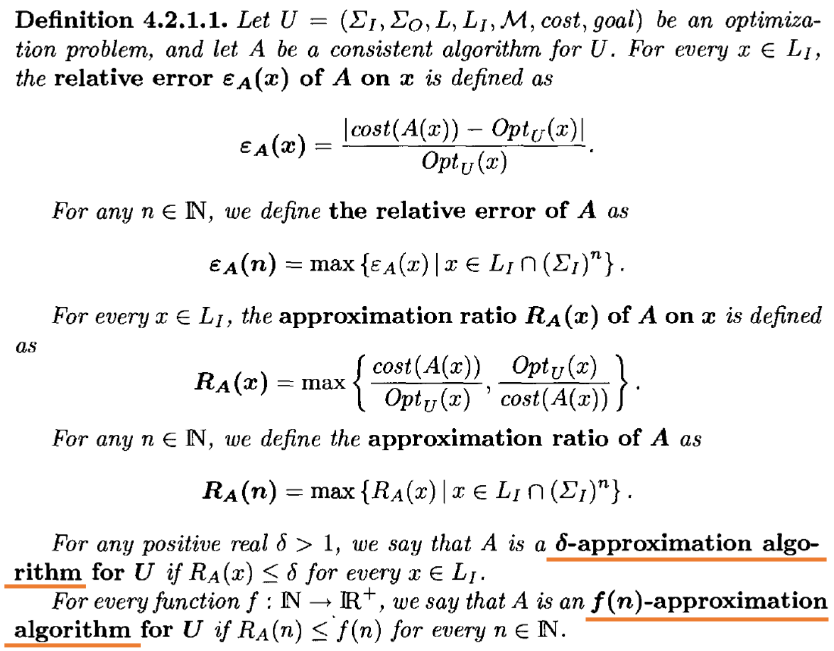
For every $x \in L_I$*, the* **approximation ratio** $\boldsymbol{R_A(x)}$ **of** $\boldsymbol{A}$ **on** $\boldsymbol{x}$ *is defined as*

$$\boldsymbol{R_A(x)} = \max\left\{\frac{cost(A(x))}{Opt_U(x)}, \frac{Opt_U(x)}{cost(A(x))}\right\}.$$

For any $n \in \mathbb{N}$*, we define the* **approximation ratio of** $\boldsymbol{A}$ *as*

$$\boldsymbol{R_A(n)} = \max\{R_A(x) \,|\, x \in L_I \cap (\Sigma_I)^n\}.$$

你能解释这些概念吗？



Definition 4.2.1.1. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *be an optimization problem, and let* A *be a consistent algorithm for* U. *For every* $x \in L_I$, *the* **relative error** $\boldsymbol{\varepsilon_A(x)}$ **of** $\boldsymbol{A}$ **on** $\boldsymbol{x}$ *is defined as*

$$\varepsilon_A(x) = \frac{|cost(A(x)) - Opt_U(x)|}{Opt_U(x)}.$$

For any $n \in \mathbb{N}$, *we define* **the relative error of** $\boldsymbol{A}$ *as*

$$\varepsilon_A(n) = \max\{\varepsilon_A(x) \,|\, x \in L_I \cap (\Sigma_I)^n\}.$$

For every $x \in L_I$, *the* **approximation ratio** $\boldsymbol{R_A(x)}$ **of** $\boldsymbol{A}$ **on** $\boldsymbol{x}$ *is defined as*

$$R_A(x) = \max\left\{\frac{cost(A(x))}{Opt_U(x)}, \frac{Opt_U(x)}{cost(A(x))}\right\}.$$

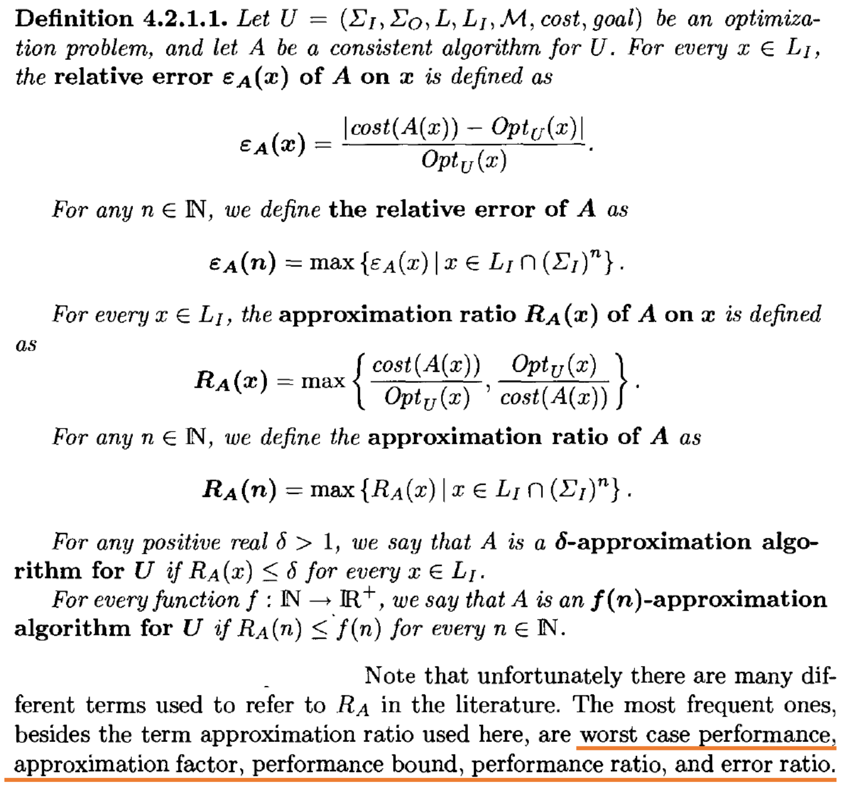
For any $n \in \mathbb{N}$, *we define the* **approximation ratio of** $\boldsymbol{A}$ *as*

$$R_A(n) = \max\{R_A(x) \,|\, x \in L_I \cap (\Sigma_I)^n\}.$$

For any positive real $\delta > 1$, *we say that* A *is a* $\boldsymbol{\delta}$**-approximation algorithm for** $\boldsymbol{U}$ *if* $R_A(x) \leq \delta$ *for every* $x \in L_I$.

For every function $f : \mathbb{N} \to \mathbb{R}^+$, *we say that* A *is an* $\boldsymbol{f(n)}$**-approximation algorithm for** $\boldsymbol{U}$ *if* $R_A(n) \leq f(n)$ *for every* $n \in \mathbb{N}$.

你能解释这些概念吗？



Definition 4.2.1.1. *Let $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ be an optimization problem, and let A be a consistent algorithm for U. For every $x \in L_I$, the* **relative error $\varepsilon_A(x)$ of A on x** *is defined as*

$$\boldsymbol{\varepsilon_A(x)} = \frac{|cost(A(x)) - Opt_U(x)|}{Opt_U(x)}.$$

For any $n \in \mathbb{N}$, we define **the relative error of A** *as*

$$\boldsymbol{\varepsilon_A(n)} = \max\left\{\varepsilon_A(x) \,|\, x \in L_I \cap (\Sigma_I)^n\right\}.$$

For every $x \in L_I$, the **approximation ratio $R_A(x)$ of A on x** *is defined as*

$$\boldsymbol{R_A(x)} = \max\left\{\frac{cost(A(x))}{Opt_U(x)}, \frac{Opt_U(x)}{cost(A(x))}\right\}.$$

For any $n \in \mathbb{N}$, we define the **approximation ratio of A** *as*

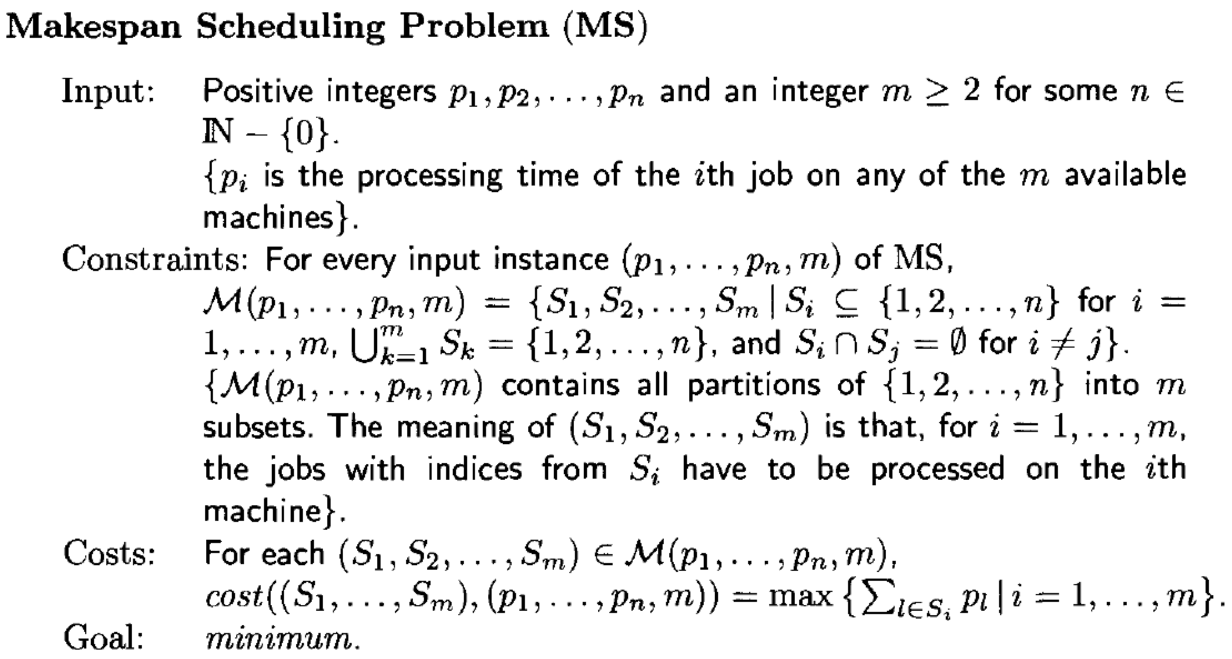
$$\boldsymbol{R_A(n)} = \max\left\{R_A(x) \,|\, x \in L_I \cap (\Sigma_I)^n\right\}.$$

For any positive real $\delta > 1$, we say that A is a **δ-approximation algorithm for U** *if $R_A(x) \leq \delta$ for every $x \in L_I$.*

For every function $f : \mathbb{N} \to \mathbb{R}^+$, we say that A is an **$f(n)$-approximation algorithm for U** *if $R_A(n) \leq f(n)$ for every $n \in \mathbb{N}$.*

Note that unfortunately there are many different terms used to refer to R_A in the literature. The most frequent ones, besides the term approximation ratio used here, are worst case performance, approximation factor, performance bound, performance ratio, and error ratio.

你还记得这个问题吗？



Makespan Scheduling Problem (MS)

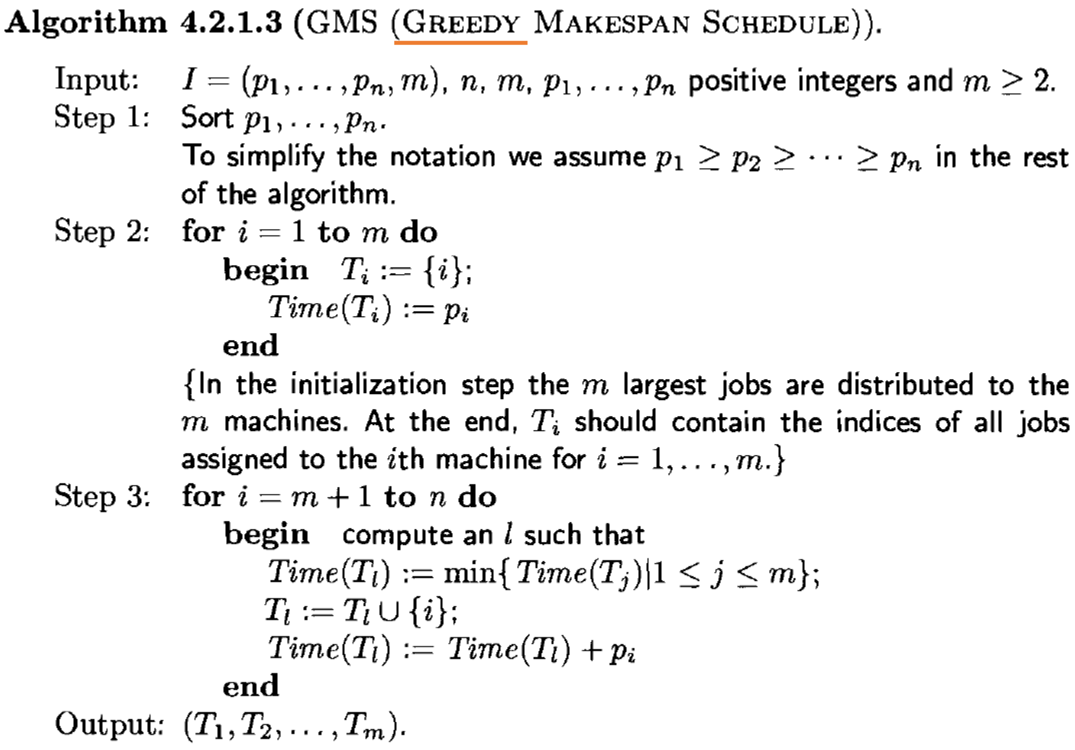
Input: Positive integers $p_1, p_2, \ldots, p_n$ and an integer $m \geq 2$ for some $n \in \mathbb{N} - \{0\}$.
$\{p_i$ is the processing time of the ith job on any of the m available machines$\}$.

Constraints: For every input instance $(p_1, \ldots, p_n, m)$ of MS,
$\mathcal{M}(p_1, \ldots, p_n, m) = \{S_1, S_2, \ldots, S_m \mid S_i \subseteq \{1, 2, \ldots, n\}$ for $i = 1, \ldots, m$, $\bigcup_{k=1}^{m} S_k = \{1, 2, \ldots, n\}$, and $S_i \cap S_j = \emptyset$ for $i \neq j\}$.
$\{\mathcal{M}(p_1, \ldots, p_n, m)$ contains all partitions of $\{1, 2, \ldots, n\}$ into m subsets. The meaning of $(S_1, S_2, \ldots, S_m)$ is that, for $i = 1, \ldots, m$, the jobs with indices from S_i have to be processed on the ith machine$\}$.

Costs: For each $(S_1, S_2, \ldots, S_m) \in \mathcal{M}(p_1, \ldots, p_n, m)$,
$cost((S_1, \ldots, S_m), (p_1, \ldots, p_n, m)) = \max\left\{\sum_{l \in S_i} p_l \mid i = 1, \ldots, m\right\}$.

Goal: *minimum*.

你理解这个算法了吗？



Algorithm 4.2.1.3 (GMS (GREEDY MAKESPAN SCHEDULE)).

```
Input:   $I = (p_1, \ldots, p_n, m)$, $n$, $m$, $p_1, \ldots, p_n$ positive integers and $m \geq 2$.
Step 1:  Sort $p_1, \ldots, p_n$.
         To simplify the notation we assume $p_1 \geq p_2 \geq \cdots \geq p_n$ in the rest
         of the algorithm.
Step 2:  for $i = 1$ to $m$ do
             begin  $T_i := \{i\}$;
                 $Time(T_i) := p_i$
             end
         {In the initialization step the $m$ largest jobs are distributed to the
         $m$ machines. At the end, $T_i$ should contain the indices of all jobs
         assigned to the $i$th machine for $i = 1, \ldots, m$.}
Step 3:  for $i = m + 1$ to $n$ do
             begin  compute an $l$ such that
                 $Time(T_l) := \min\{Time(T_j) | 1 \leq j \leq m\}$;
                 $T_l := T_l \cup \{i\}$;
                 $Time(T_l) := Time(T_l) + p_i$
             end
Output:  $(T_1, T_2, \ldots, T_m)$.
```

你理解这段证明了吗？

$$Opt_{\mathrm{MS}}(I) \geq p_1 \geq p_2 \geq \cdots \geq p_n. \tag{4.1}$$

$$Opt_{\mathrm{MS}}(I) \geq \frac{\sum_{i=1}^{n} p_i}{m} \tag{4.2}$$

$$p_k \leq \frac{\sum_{i=1}^{k} p_i}{k} \tag{4.3}$$

(1) Let $n \leq m$.
Since $Opt_{\mathrm{MS}}(I) \geq p_1$ (4.1) and $cost(\{1\},\{2\},\dots,\{n\},\emptyset,\dots,\emptyset) = p_1$, GMS has found an optimal solution and so the approximation ratio is 1.

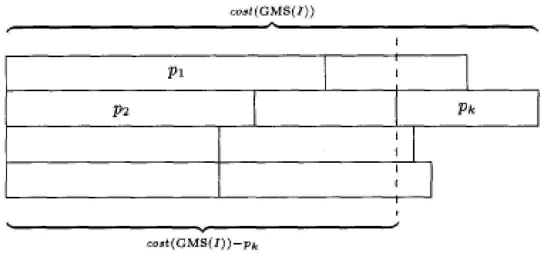
(2) Let $n > m$.
Let T_l be such that $cost(T_l) = \sum_{r \in T_l} p_r = cost(\mathrm{GMS}(I))$, and let k be the largest index in T_l. If $k \leq m$, then $|T_l| = 1$ and so $Opt_{\mathrm{MS}}(I) = p_1 = p_k$ and $\mathrm{GMS}(I)$ is an optimal solution.
Now, assume $m < k$. Following Figure 4.2 we see that

$$Opt_{\mathrm{MS}}(I) \geq cost(\mathrm{GMS}(I)) - p_k \tag{4.4}$$

because of $\sum_{i=1}^{k-1} p_i \geq m \cdot [cost(\mathrm{GMS}(I)) - p_k]$ and (4.2).

$$cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I) \underset{(4.4)}{\leq} p_k \underset{(4.3)}{\leq} \left(\sum_{i=1}^{k} p_i\right) \Big/ k. \tag{4.5}$$

$$\frac{cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I)}{Opt_{\mathrm{MS}}(I)} \underset{\substack{(4.5)\\(4.2)}}{\leq} \frac{\left(\sum_{i=1}^{k} p_i\right)/k}{\left(\sum_{i=1}^{n} p_i\right)/m} \leq \frac{m}{k} < 1.$$



你理解这段证明了吗？

$$Opt_{\mathrm{MS}}(I) \geq p_1 \geq p_2 \geq \cdots \geq p_n. \qquad (4.1)$$

$$Opt_{\mathrm{MS}}(I) \geq \frac{\sum_{i=1}^{n} p_i}{m} \qquad (4.2)$$

$$p_k \leq \frac{\sum_{i=1}^{k} p_i}{k} \qquad (4.3)$$

(1) Let $n \leq m$.
Since $Opt_{\mathrm{MS}}(I) \geq p_1$ (4.1) and $cost(\{1\},\{2\},\ldots,\{n\},\emptyset,\ldots,\emptyset) = p_1$, GMS has found an optimal solution and so the approximation ratio is 1.

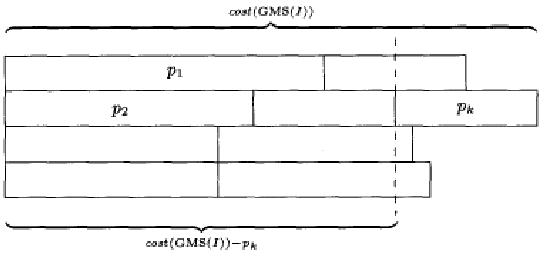
(2) Let $n > m$.
Let T_l be such that $cost(T_l) = \sum_{r \in T_l} p_r = cost(\mathrm{GMS}(I))$, and let k be the largest index in T_l. If $k \leq m$, then $|T_l| = 1$ and so $Opt_{\mathrm{MS}}(I) = p_1 = p_k$ and $\mathrm{GMS}(I)$ is an optimal solution.
Now, assume $m < k$. Following Figure 4.2 we see that

$$Opt_{\mathrm{MS}}(I) \geq cost(\mathrm{GMS}(I)) - p_k \qquad (4.4)$$

because of $\sum_{i=1}^{k-1} p_i \geq m \cdot [cost(\mathrm{GMS}(I)) - p_k]$ and (4.2).

$$cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I) \underset{(4.4)}{\leq} p_k \underset{(4.3)}{\leq} \left(\sum_{i=1}^{k} p_i\right) \Big/ k. \qquad (4.5)$$

$$\frac{cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I)}{Opt_{\mathrm{MS}}(I)} \underset{\substack{(4.5)\\(4.2)}}{\leq} \frac{\left(\sum_{i=1}^{k} p_i\right)/k}{\left(\sum_{i=1}^{n} p_i\right)/m} \leq \frac{m}{k} < 1.$$



你理解这段证明了吗？

$$Opt_{\text{MS}}(I) \geq p_1 \geq p_2 \geq \cdots \geq p_n. \tag{4.1}$$

$$Opt_{\text{MS}}(I) \geq \frac{\sum_{i=1}^{n} p_i}{m} \tag{4.2}$$

$$p_k \leq \frac{\sum_{i=1}^{k} p_i}{k} \tag{4.3}$$

(1) Let $n \leq m$.
Since $Opt_{\text{MS}}(I) \geq p_1$ (4.1) and $cost(\{1\},\{2\},\dots,\{n\},\emptyset,\dots,\emptyset) = p_1$, GMS has found an optimal solution and so the approximation ratio is 1.

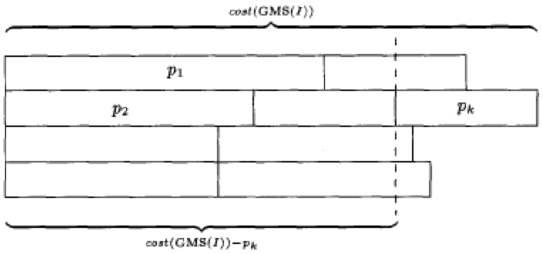
(2) Let $n > m$.
Let T_l be such that $cost(T_l) = \sum_{r\in T_l} p_r = cost(\text{GMS}(I))$, and let k be the largest index in T_l. If $k \leq m$, then $|T_l| = 1$ and so $Opt_{\text{MS}}(I) = p_1 = p_k$ and GMS(I) is an optimal solution.
Now, assume $m < k$. Following Figure 4.2 we see that

$$Opt_{\text{MS}}(I) \geq cost(\text{GMS}(I)) - p_k \tag{4.4}$$

because of $\sum_{i=1}^{k-1} p_i \geq m \cdot [cost(\text{GMS}(I)) - p_k]$ and (4.2).

$$cost(\text{GMS}(I)) - Opt_{\text{MS}}(I) \underset{(4.4)}{\leq} p_k \underset{(4.3)}{\leq} \left(\sum_{i=1}^{k} p_i\right) \Big/ k. \tag{4.5}$$

$$\frac{cost(\text{GMS}(I)) - Opt_{\text{MS}}(I)}{Opt_{\text{MS}}(I)} \underset{\substack{(4.5)\\(4.2)}}{\leq} \frac{\left(\sum_{i=1}^{k} p_i\right)/k}{\left(\sum_{i=1}^{n} p_i\right)/m} \leq \frac{m}{k} < 1.$$



你理解这段证明了吗？

$$Opt_{\mathrm{MS}}(I) \geq p_1 \geq p_2 \geq \cdots \geq p_n. \qquad (4.1)$$

$$Opt_{\mathrm{MS}}(I) \geq \frac{\sum_{i=1}^{n} p_i}{m} \qquad (4.2)$$

$$p_k \leq \frac{\sum_{i=1}^{k} p_i}{k} \qquad (4.3)$$

(1) Let $n \leq m$.
Since $Opt_{\mathrm{MS}}(I) \geq p_1$ (4.1) and $cost(\{1\},\{2\},\ldots,\{n\},\emptyset,\ldots,\emptyset) = p_1$, GMS has found an optimal solution and so the approximation ratio is 1.

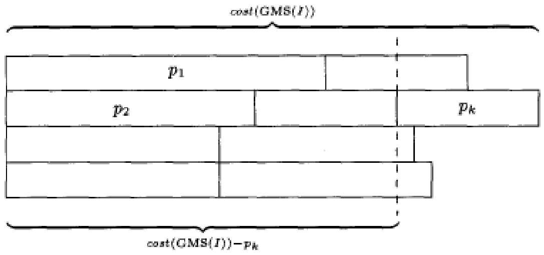
(2) Let $n > m$.
Let T_l be such that $cost(T_l) = \sum_{r \in T_l} p_r = cost(\mathrm{GMS}(I))$, and let k be the largest index in T_l. If $k \leq m$, then $|T_l| = 1$ and so $Opt_{\mathrm{MS}}(I) = p_1 = p_k$ and $\mathrm{GMS}(I)$ is an optimal solution.
Now, assume $m < k$. Following Figure 4.2 we see that

$$Opt_{\mathrm{MS}}(I) \geq cost(\mathrm{GMS}(I)) - p_k \qquad (4.4)$$

because of $\sum_{i=1}^{k-1} p_i \geq m \cdot [cost(\mathrm{GMS}(I)) - p_k]$ and (4.2).

$$cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I) \underset{(4.4)}{\leq} p_k \underset{(4.3)}{\leq} \left(\sum_{i=1}^{k} p_i\right) \Big/ k. \qquad (4.5)$$

$$\frac{cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I)}{Opt_{\mathrm{MS}}(I)} \underset{\substack{(4.5)\\(4.2)}}{\leq} \frac{\left(\sum_{i=1}^{k} p_i\right)/k}{\left(\sum_{i=1}^{n} p_i\right)/m} \leq \frac{m}{k} < 1.$$



你理解这段证明了吗？

$$Opt_{\mathrm{MS}}(I) \geq p_1 \geq p_2 \geq \cdots \geq p_n. \qquad (4.1)$$

$$Opt_{\mathrm{MS}}(I) \geq \frac{\sum_{i=1}^{n} p_i}{m} \qquad (4.2)$$

$$p_k \leq \frac{\sum_{i=1}^{k} p_i}{k} \qquad (4.3)$$

(1) Let $n \leq m$.
Since $Opt_{\mathrm{MS}}(I) \geq p_1$ (4.1) and $cost(\{1\},\{2\},\ldots,\{n\},\emptyset,\ldots,\emptyset) = p_1$, GMS has found an optimal solution and so the approximation ratio is 1.

(2) Let $n > m$.
Let T_l be such that $cost(T_l) = \sum_{r \in T_l} p_r = cost(\mathrm{GMS}(I))$, and let k be the largest index in T_l. If $k \leq m$, then $|T_l| = 1$ and so $Opt_{\mathrm{MS}}(I) = p_1 = p_k$ and GMS(I) is an optimal solution.

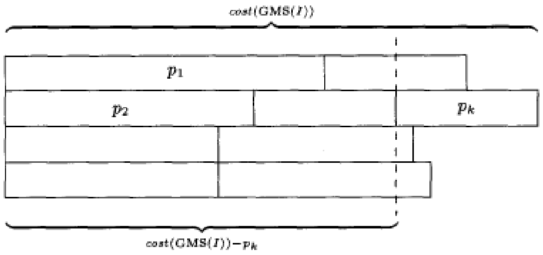
Now, assume $m < k$. Following Figure 4.2 we see that

$$Opt_{\mathrm{MS}}(I) \geq cost(\mathrm{GMS}(I)) - p_k \qquad (4.4)$$

because of $\sum_{i=1}^{k-1} p_i \geq m \cdot [cost(\mathrm{GMS}(I)) - p_k]$ and (4.2).

$$cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I) \underset{(4.4)}{\leq} p_k \underset{(4.3)}{\leq} \left(\sum_{i=1}^{k} p_i\right) \Big/ k. \qquad (4.5)$$

$$\frac{cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I)}{Opt_{\mathrm{MS}}(I)} \underset{(4.2)}{\underset{(4.5)}{\leq}} \frac{\left(\sum_{i=1}^{k} p_i\right)/k}{\left(\sum_{i=1}^{n} p_i\right)/m} \leq \frac{m}{k} < 1.$$



你理解这段证明了吗？

$$Opt_{\mathrm{MS}}(I) \geq p_1 \geq p_2 \geq \cdots \geq p_n. \qquad (4.1)$$

$$Opt_{\mathrm{MS}}(I) \geq \frac{\sum_{i=1}^{n} p_i}{m} \qquad (4.2)$$

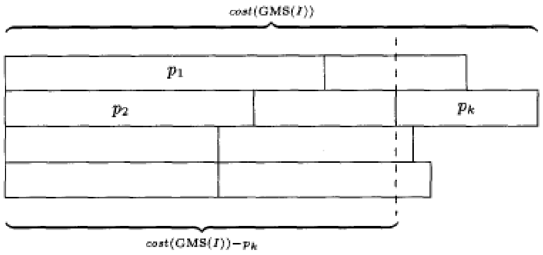
$$p_k \leq \frac{\sum_{i=1}^{k} p_i}{k} \qquad (4.3)$$

(1) Let $n \leq m$.
Since $Opt_{\mathrm{MS}}(I) \geq p_1$ (4.1) and $cost(\{1\},\{2\},\ldots,\{n\},\emptyset,\ldots,\emptyset) = p_1$, GMS has found an optimal solution and so the approximation ratio is 1.

(2) Let $n > m$.
Let T_l be such that $cost(T_l) = \sum_{r \in T_l} p_r = cost(\mathrm{GMS}(I))$, and let k be the largest index in T_l. If $k \leq m$, then $|T_l| = 1$ and so $Opt_{\mathrm{MS}}(I) = p_1 = p_k$ and GMS(I) is an optimal solution.
Now, assume $m < k$. Following Figure 4.2 we see that

$$\underline{Opt_{\mathrm{MS}}(I) \geq cost(\mathrm{GMS}(I)) - p_k} \qquad (4.4)$$

because of $\sum_{i=1}^{k-1} p_i \geq m \cdot [cost(\mathrm{GMS}(I)) - p_k]$ and (4.2).



$$cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I) \underset{(4.4)}{\leq} p_k \underset{(4.3)}{\leq} \left(\sum_{i=1}^{k} p_i\right) \Big/ k. \qquad (4.5)$$

$$\frac{cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I)}{Opt_{\mathrm{MS}}(I)} \underset{\substack{(4.5)\\(4.2)}}{\leq} \frac{\left(\sum_{i=1}^{k} p_i\right)/k}{\left(\sum_{i=1}^{n} p_i\right)/m} \leq \frac{m}{k} < 1.$$

你理解这段证明了吗？

$$Opt_{\mathrm{MS}}(I) \geq p_1 \geq p_2 \geq \cdots \geq p_n. \qquad (4.1)$$

$$Opt_{\mathrm{MS}}(I) \geq \frac{\sum_{i=1}^{n} p_i}{m} \qquad (4.2)$$

$$p_k \leq \frac{\sum_{i=1}^{k} p_i}{k} \qquad (4.3)$$

(1) Let $n \leq m$.
Since $Opt_{\mathrm{MS}}(I) \geq p_1$ (4.1) and $cost(\{1\}, \{2\}, \ldots, \{n\}, \emptyset, \ldots, \emptyset) = p_1$, GMS has found an optimal solution and so the approximation ratio is 1.

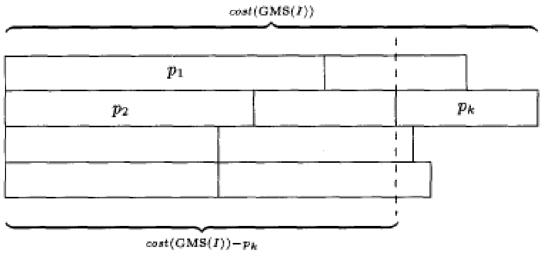
(2) Let $n > m$.
Let T_l be such that $cost(T_l) = \sum_{r \in T_l} p_r = cost(\mathrm{GMS}(I))$, and let k be the largest index in T_l. If $k \leq m$, then $|T_l| = 1$ and so $Opt_{\mathrm{MS}}(I) = p_1 = p_k$ and $\mathrm{GMS}(I)$ is an optimal solution.
Now, assume $m < k$. Following Figure 4.2 we see that

$$Opt_{\mathrm{MS}}(I) \geq cost(\mathrm{GMS}(I)) - p_k \qquad (4.4)$$

because of $\sum_{i=1}^{k-1} p_i \geq m \cdot [cost(\mathrm{GMS}(I)) - p_k]$ and (4.2).

$$cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I) \underset{(4.4)}{\leq} p_k \underset{(4.3)}{\leq} \left(\sum_{i=1}^{k} p_i\right) \Big/ k. \qquad (4.5)$$

$$\frac{cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I)}{Opt_{\mathrm{MS}}(I)} \underset{\substack{(4.5)\\(4.2)}}{\leq} \frac{\left(\sum_{i=1}^{k} p_i\right)/k}{\left(\sum_{i=1}^{n} p_i\right)/m} \leq \frac{m}{k} < 1.$$



你理解这段证明了吗？

$$Opt_{\mathrm{MS}}(I) \geq p_1 \geq p_2 \geq \cdots \geq p_n. \qquad (4.1)$$

$$Opt_{\mathrm{MS}}(I) \geq \frac{\sum_{i=1}^{n} p_i}{m} \qquad (4.2)$$

$$p_k \leq \frac{\sum_{i=1}^{k} p_i}{k} \qquad (4.3)$$

(1) Let $n \leq m$.
Since $Opt_{\mathrm{MS}}(I) \geq p_1$ (4.1) and $cost(\{1\},\{2\},\ldots,\{n\},\emptyset,\ldots,\emptyset) = p_1$, GMS has found an optimal solution and so the approximation ratio is 1.

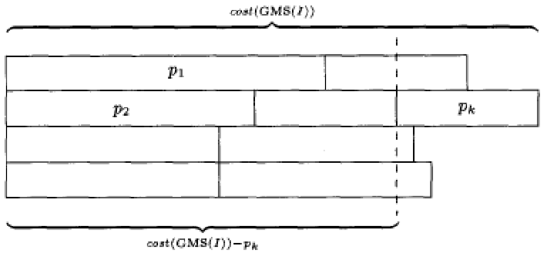
(2) Let $n > m$.
Let T_l be such that $cost(T_l) = \sum_{r \in T_l} p_r = cost(\mathrm{GMS}(I))$, and let k be the largest index in T_l. If $k \leq m$, then $|T_l| = 1$ and so $Opt_{\mathrm{MS}}(I) = p_1 = p_k$ and GMS(I) is an optimal solution.
Now, assume $m < k$. Following Figure 4.2 we see that

$$Opt_{\mathrm{MS}}(I) \geq cost(\mathrm{GMS}(I)) - p_k \qquad (4.4)$$

because of $\sum_{i=1}^{k-1} p_i \geq m \cdot [cost(\mathrm{GMS}(I)) - p_k]$ and (4.2).

$$cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I) \underset{(4.4)}{\leq} p_k \underset{(4.3)}{\leq} \left(\sum_{i=1}^{k} p_i\right) \Big/ k. \qquad (4.5)$$

$$\frac{cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I)}{Opt_{\mathrm{MS}}(I)} \underset{\substack{(4.5)\\(4.2)}}{\leq} \frac{\left(\sum_{i=1}^{k} p_i\right)/k}{\left(\sum_{i=1}^{n} p_i\right)/m} \leq \frac{m}{k} < 1.$$



这个算法的近似比是多少？好不好？

$$Opt_{\mathrm{MS}}(I) \geq p_1 \geq p_2 \geq \cdots \geq p_n. \qquad (4.1)$$

$$Opt_{\mathrm{MS}}(I) \geq \frac{\sum_{i=1}^{n} p_i}{m} \qquad (4.2)$$

$$p_k \leq \frac{\sum_{i=1}^{k} p_i}{k} \qquad (4.3)$$

(1) Let $n \leq m$.
Since $Opt_{\mathrm{MS}}(I) \geq p_1$ (4.1) and $cost(\{1\},\{2\},\ldots,\{n\},\emptyset,\ldots,\emptyset) = p_1$, GMS has found an optimal solution and so the approximation ratio is 1.

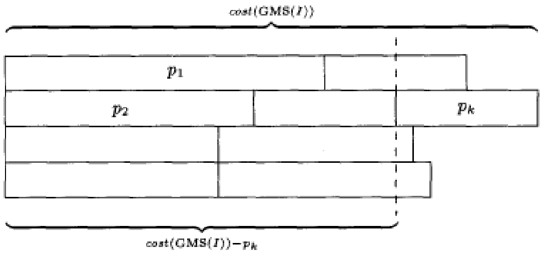
(2) Let $n > m$.
Let T_l be such that $cost(T_l) = \sum_{r \in T_l} p_r = cost(\mathrm{GMS}(I))$, and let k be the largest index in T_l. If $k \leq m$, then $|T_l| = 1$ and so $Opt_{\mathrm{MS}}(I) = p_1 = p_k$ and GMS(I) is an optimal solution.
Now, assume $m < k$. Following Figure 4.2 we see that

$$Opt_{\mathrm{MS}}(I) \geq cost(\mathrm{GMS}(I)) - p_k \qquad (4.4)$$

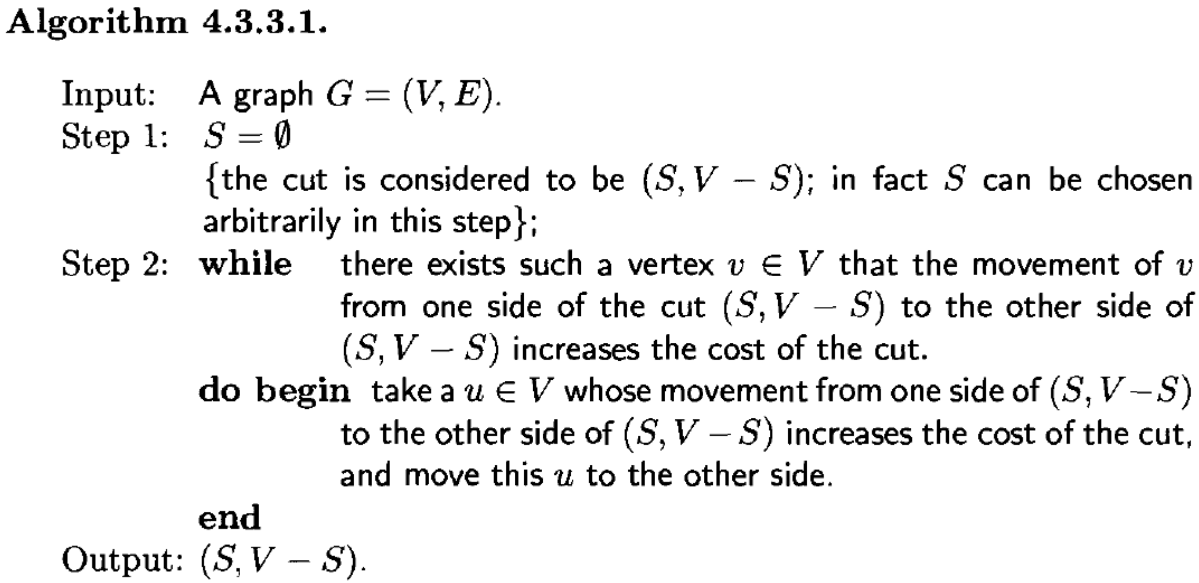
because of $\sum_{i=1}^{k-1} p_i \geq m \cdot [cost(\mathrm{GMS}(I)) - p_k]$ and (4.2).

$$cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I) \underset{(4.4)}{\leq} p_k \underset{(4.3)}{\leq} \left(\sum_{i=1}^{k} p_i\right) \Big/ k. \qquad (4.5)$$

$$\frac{cost(\mathrm{GMS}(I)) - Opt_{\mathrm{MS}}(I)}{Opt_{\mathrm{MS}}(I)} \underset{\substack{(4.5)\\(4.2)}}{\leq} \frac{\left(\sum_{i=1}^{k} p_i\right)/k}{\left(\sum_{i=1}^{n} p_i\right)/m} \leq \frac{m}{k} < 1.$$



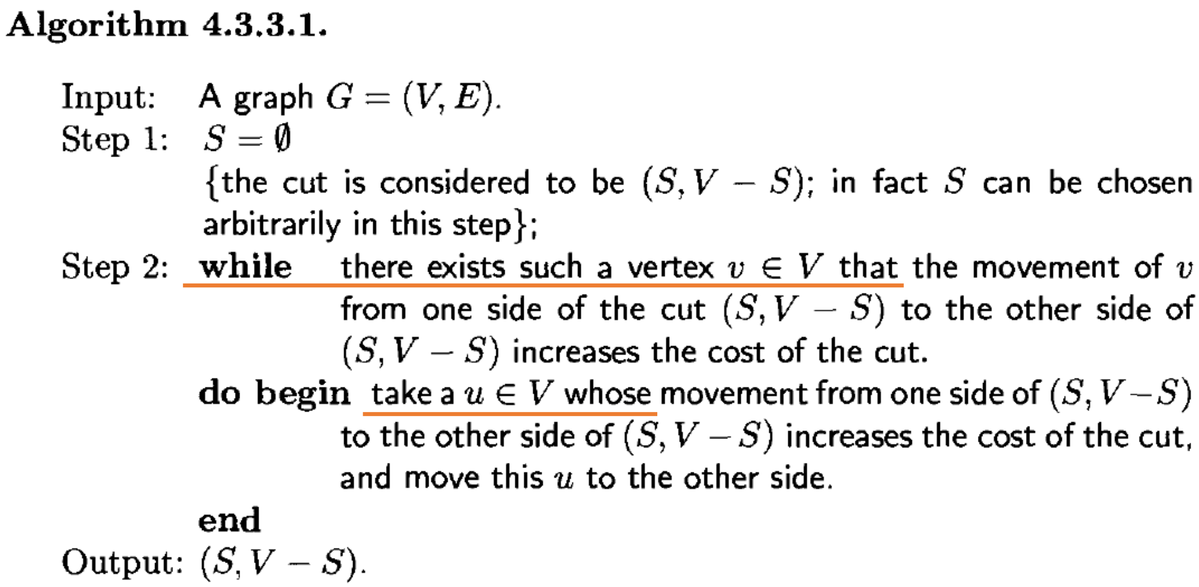
你理解这个算法了吗？



Algorithm 4.3.3.1.

```
Input:   A graph $G = (V, E)$.
Step 1:  $S = \emptyset$
         {the cut is considered to be $(S, V - S)$; in fact $S$ can be chosen
         arbitrarily in this step};
Step 2:  while    there exists such a vertex $v \in V$ that the movement of $v$
                  from one side of the cut $(S, V - S)$ to the other side of
                  $(S, V - S)$ increases the cost of the cut.
         do begin take a $u \in V$ whose movement from one side of $(S, V - S)$
                  to the other side of $(S, V - S)$ increases the cost of the cut,
                  and move this $u$ to the other side.
         end
Output:  $(S, V - S)$.
```

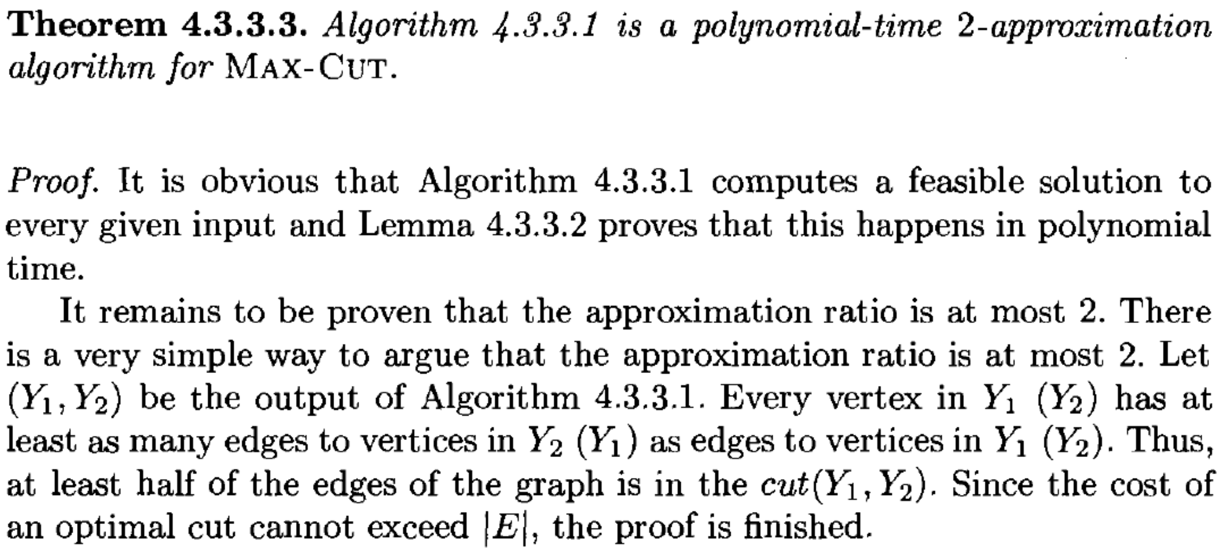
你能解释这个算法的时间复杂度吗？



Algorithm 4.3.3.1.

```
Input:   A graph $G = (V, E)$.
Step 1:  $S = \emptyset$
         {the cut is considered to be $(S, V - S)$; in fact $S$ can be chosen
         arbitrarily in this step};
Step 2:  while  there exists such a vertex $v \in V$ that the movement of $v$
                from one side of the cut $(S, V - S)$ to the other side of
                $(S, V - S)$ increases the cost of the cut.
         do begin  take a $u \in V$ whose movement from one side of $(S, V - S)$
                to the other side of $(S, V - S)$ increases the cost of the cut,
                and move this $u$ to the other side.
         end
Output:  $(S, V - S)$.
```

你理解这段证明了吗？

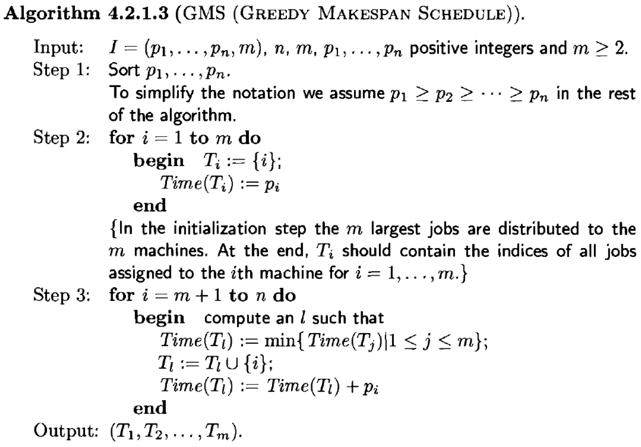


Theorem 4.3.3.3. *Algorithm 4.3.3.1 is a polynomial-time* 2*-approximation algorithm for* MAX-CUT.

Proof. It is obvious that Algorithm 4.3.3.1 computes a feasible solution to every given input and Lemma 4.3.3.2 proves that this happens in polynomial time.

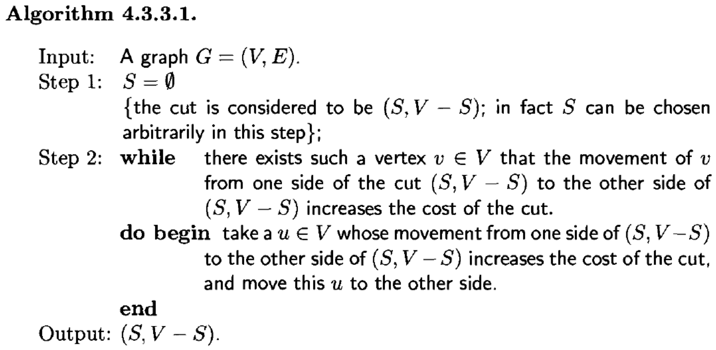
It remains to be proven that the approximation ratio is at most 2. There is a very simple way to argue that the approximation ratio is at most 2. Let (Y_1, Y_2) be the output of Algorithm 4.3.3.1. Every vertex in Y_1 (Y_2) has at least as many edges to vertices in Y_2 (Y_1) as edges to vertices in Y_1 (Y_2). Thus, at least half of the edges of the graph is in the $cut(Y_1, Y_2)$. Since the cost of an optimal cut cannot exceed $|E|$, the proof is finished.

Greedy和Local Search有什么区别?



Algorithm 4.2.1.3 (GMS (Greedy Makespan Schedule)).

```
Input:   $I = (p_1, \ldots, p_n, m)$, $n$, $m$, $p_1, \ldots, p_n$ positive integers and $m \geq 2$.
Step 1:  Sort $p_1, \ldots, p_n$.
         To simplify the notation we assume $p_1 \geq p_2 \geq \cdots \geq p_n$ in the rest
         of the algorithm.
Step 2:  for $i = 1$ to $m$ do
            begin  $T_i := \{i\}$;
               $Time(T_i) := p_i$
            end
         {In the initialization step the $m$ largest jobs are distributed to the
         $m$ machines. At the end, $T_i$ should contain the indices of all jobs
         assigned to the $i$th machine for $i = 1, \ldots, m$.}
Step 3:  for $i = m + 1$ to $n$ do
            begin  compute an $l$ such that
               $Time(T_l) := \min\{Time(T_j) | 1 \leq j \leq m\}$;
               $T_l := T_l \cup \{i\}$;
               $Time(T_l) := Time(T_l) + p_i$
            end
Output:  $(T_1, T_2, \ldots, T_m)$.
```



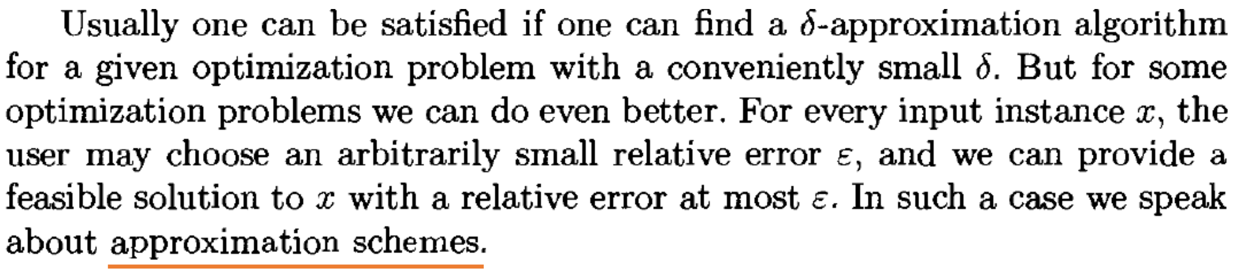
Algorithm 4.3.3.1.

```
Input:   A graph $G = (V, E)$.
Step 1:  $S = \emptyset$
         {the cut is considered to be $(S, V - S)$; in fact $S$ can be chosen
         arbitrarily in this step};
Step 2:  while  there exists such a vertex $v \in V$ that the movement of $v$
                from one side of the cut $(S, V - S)$ to the other side of
                $(S, V - S)$ increases the cost of the cut.
         do begin  take a $u \in V$ whose movement from one side of $(S, V - S)$
                to the other side of $(S, V - S)$ increases the cost of the cut,
                and move this $u$ to the other side.
         end
Output:  $(S, V - S)$.
```

Greedy和Local Search有什么区别?

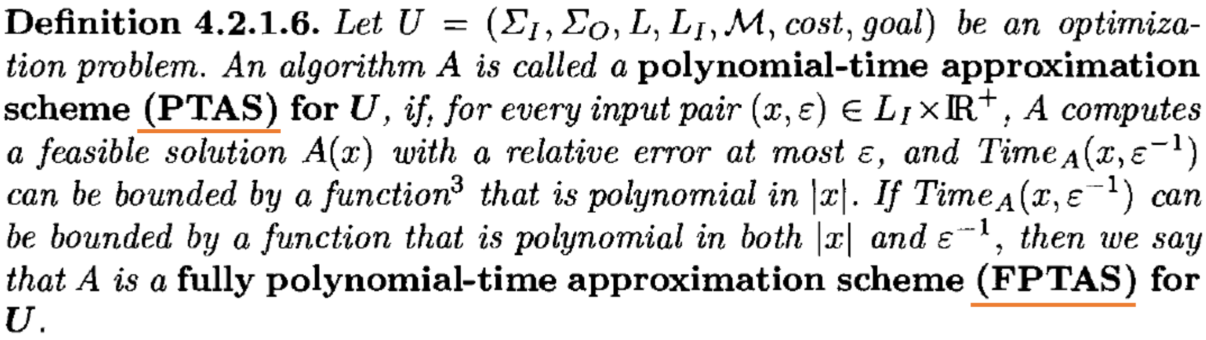
- 请分别为以下问题设计greedy和local search算法
  - MAX-SAT
  - MAX-CL
  - longest simple path

你能解释这些概念吗？



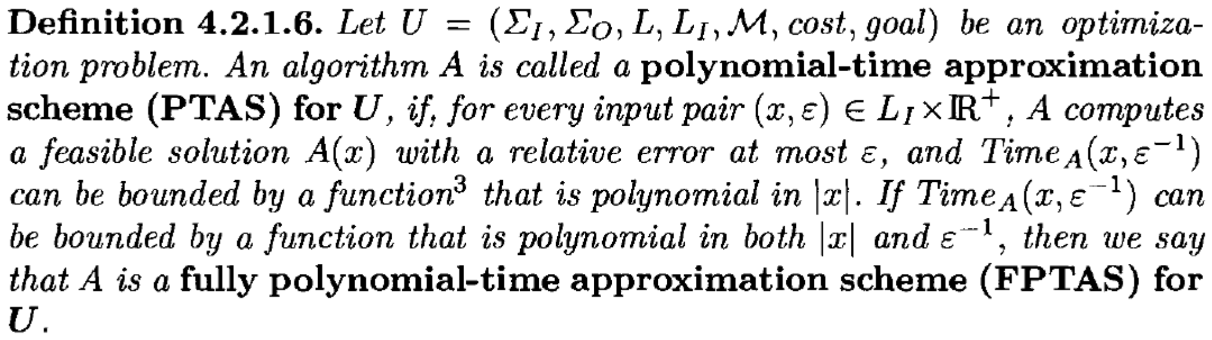
Usually one can be satisfied if one can find a δ-approximation algorithm for a given optimization problem with a conveniently small δ. But for some optimization problems we can do even better. For every input instance x, the user may choose an arbitrarily small relative error ε, and we can provide a feasible solution to x with a relative error at most ε. In such a case we speak about approximation schemes.

你能解释这些概念吗？



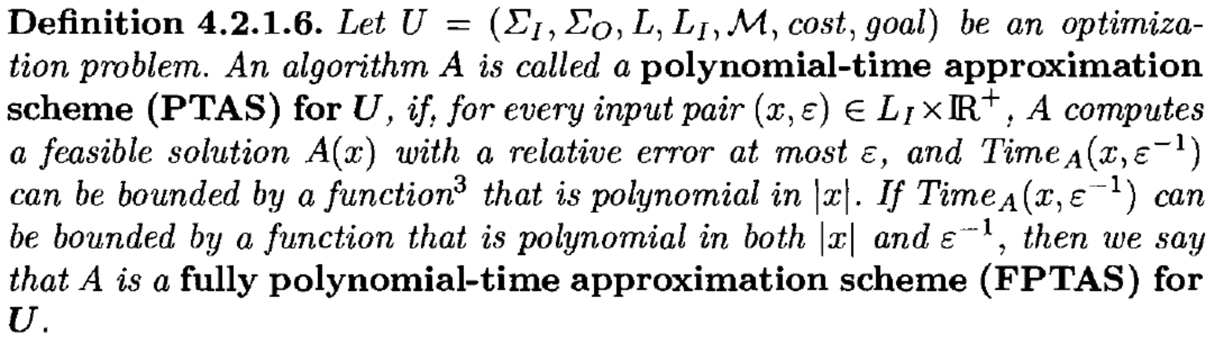
Definition 4.2.1.6. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *be an optimization problem. An algorithm A is called a* **polynomial-time approximation scheme (PTAS) for** $\boldsymbol{U}$, *if, for every input pair* $(x, \varepsilon) \in L_I \times \mathbb{R}^+$, *A computes a feasible solution* $A(x)$ *with a relative error at most* ε, *and* $Time_A(x, \varepsilon^{-1})$ *can be bounded by a function*[3] *that is polynomial in* $|x|$. *If* $Time_A(x, \varepsilon^{-1})$ *can be bounded by a function that is polynomial in both* $|x|$ *and* ε^{-1}, *then we say that A is a* **fully polynomial-time approximation scheme (FPTAS) for** $\boldsymbol{U}$.

你能解释这些概念吗？


Definition 4.2.1.6. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *be an optimization problem. An algorithm A is called a* **polynomial-time approximation scheme (PTAS) for** $\boldsymbol{U}$, *if, for every input pair* $(x, \varepsilon) \in L_I \times \mathbb{R}^+$, *A computes a feasible solution* $A(x)$ *with a relative error at most* ε, *and* $Time_A(x, \varepsilon^{-1})$ *can be bounded by a function*[3] *that is polynomial in* $|x|$. *If* $Time_A(x, \varepsilon^{-1})$ *can be bounded by a function that is polynomial in both* $|x|$ *and* ε^{-1}, *then we say that A is a* **fully polynomial-time approximation scheme (FPTAS) for** $\boldsymbol{U}$.

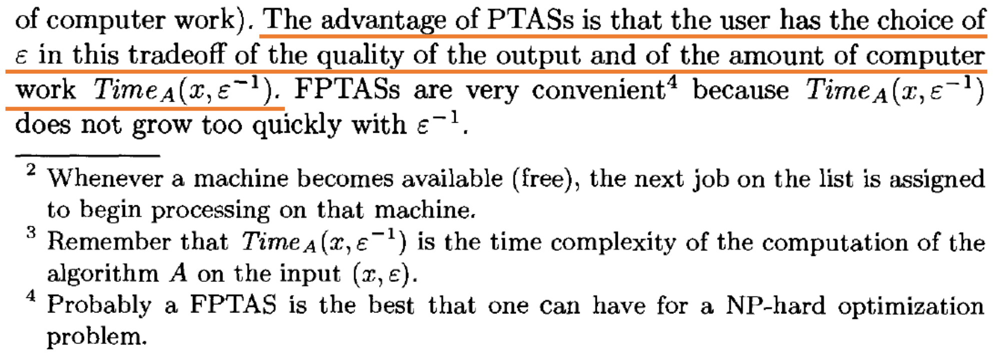
- 你能解释PTAS的意义吗？

你能解释这些概念吗？



Definition 4.2.1.6. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *be an optimization problem. An algorithm* A *is called a* **polynomial-time approximation scheme (PTAS) for** $\boldsymbol{U}$, *if, for every input pair* $(x, \varepsilon) \in L_I \times \mathbb{R}^+$, *A computes a feasible solution* $A(x)$ *with a relative error at most* ε, *and* $Time_A(x, \varepsilon^{-1})$ *can be bounded by a function*[3] *that is polynomial in* $|x|$. *If* $Time_A(x, \varepsilon^{-1})$ *can be bounded by a function that is polynomial in both* $|x|$ *and* ε^{-1}, *then we say that A is a* **fully polynomial-time approximation scheme (FPTAS) for** $\boldsymbol{U}$.

- 你能解释PTAS的意义吗？



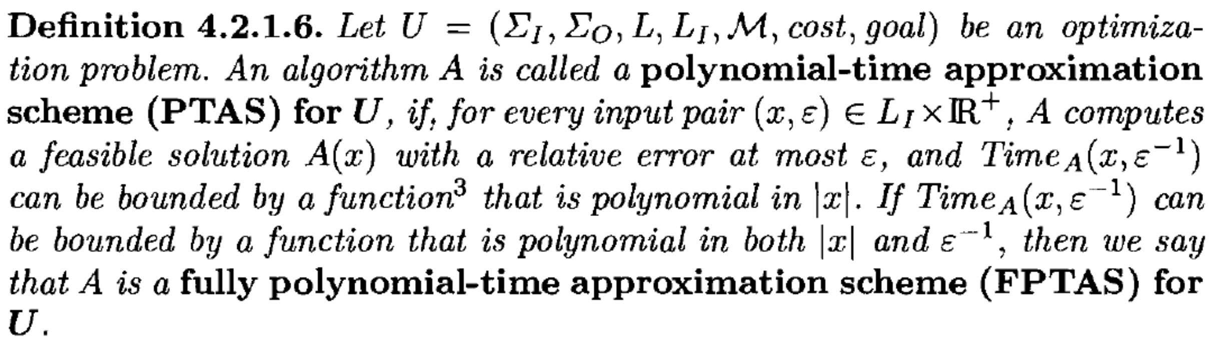
of computer work). The advantage of PTASs is that the user has the choice of ε in this tradeoff of the quality of the output and of the amount of computer work $Time_A(x, \varepsilon^{-1})$. FPTASs are very convenient[4] because $Time_A(x, \varepsilon^{-1})$ does not grow too quickly with ε^{-1}.

[2] Whenever a machine becomes available (free), the next job on the list is assigned to begin processing on that machine.

[3] Remember that $Time_A(x, \varepsilon^{-1})$ is the time complexity of the computation of the algorithm A on the input (x, ε).

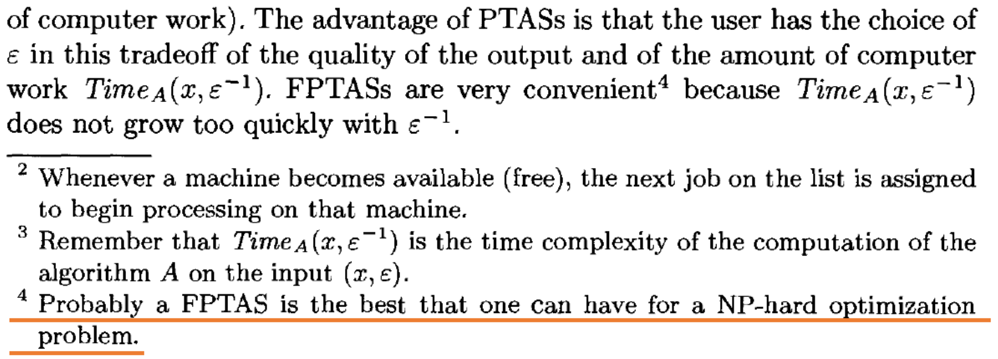
[4] Probably a FPTAS is the best that one can have for a NP-hard optimization problem.

你能解释这些概念吗?



Definition 4.2.1.6. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *be an optimization problem. An algorithm* A *is called a* **polynomial-time approximation scheme (PTAS) for** $\boldsymbol{U}$, *if, for every input pair* $(x, \varepsilon) \in L_I \times \mathbb{R}^+$, A *computes a feasible solution* $A(x)$ *with a relative error at most* ε, *and* $Time_A(x, \varepsilon^{-1})$ *can be bounded by a function*[3] *that is polynomial in* $|x|$. *If* $Time_A(x, \varepsilon^{-1})$ *can be bounded by a function that is polynomial in both* $|x|$ *and* ε^{-1}, *then we say that* A *is a* **fully polynomial-time approximation scheme (FPTAS) for** $\boldsymbol{U}$.

- 你理解这句话了吗?



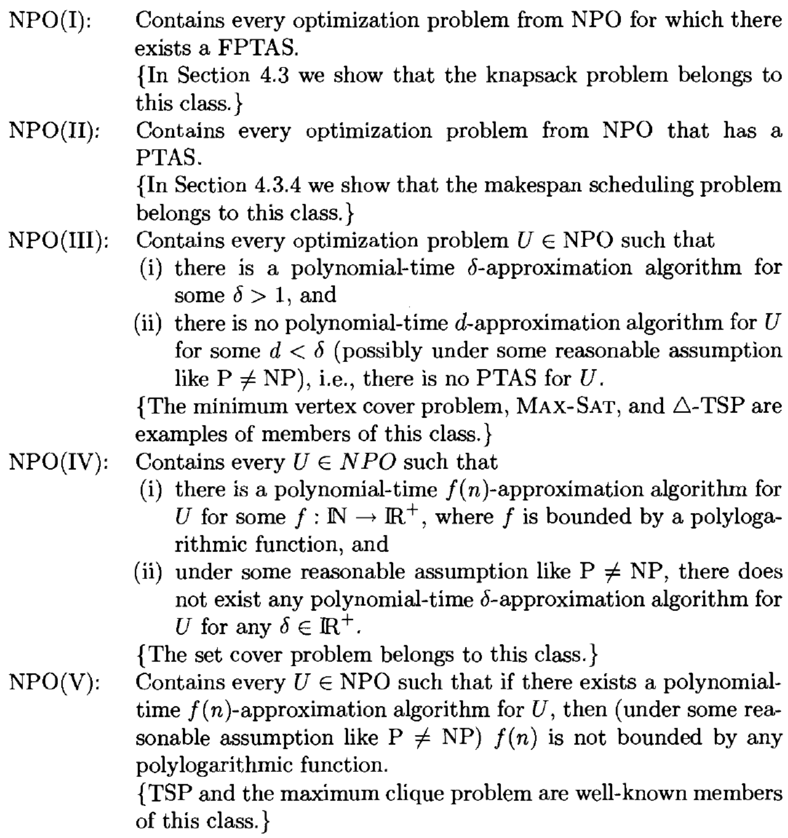
of computer work). The advantage of PTASs is that the user has the choice of ε in this tradeoff of the quality of the output and of the amount of computer work $Time_A(x, \varepsilon^{-1})$. FPTASs are very convenient[4] because $Time_A(x, \varepsilon^{-1})$ does not grow too quickly with ε^{-1}.

[2] Whenever a machine becomes available (free), the next job on the list is assigned to begin processing on that machine.

[3] Remember that $Time_A(x, \varepsilon^{-1})$ is the time complexity of the computation of the algorithm A on the input (x, ε).

[4] Probably a FPTAS is the best that one can have for a NP-hard optimization problem.

你理解这套分类体系了吗？



NPO(I): Contains every optimization problem from NPO for which there exists a FPTAS.
{In Section 4.3 we show that the knapsack problem belongs to this class.}

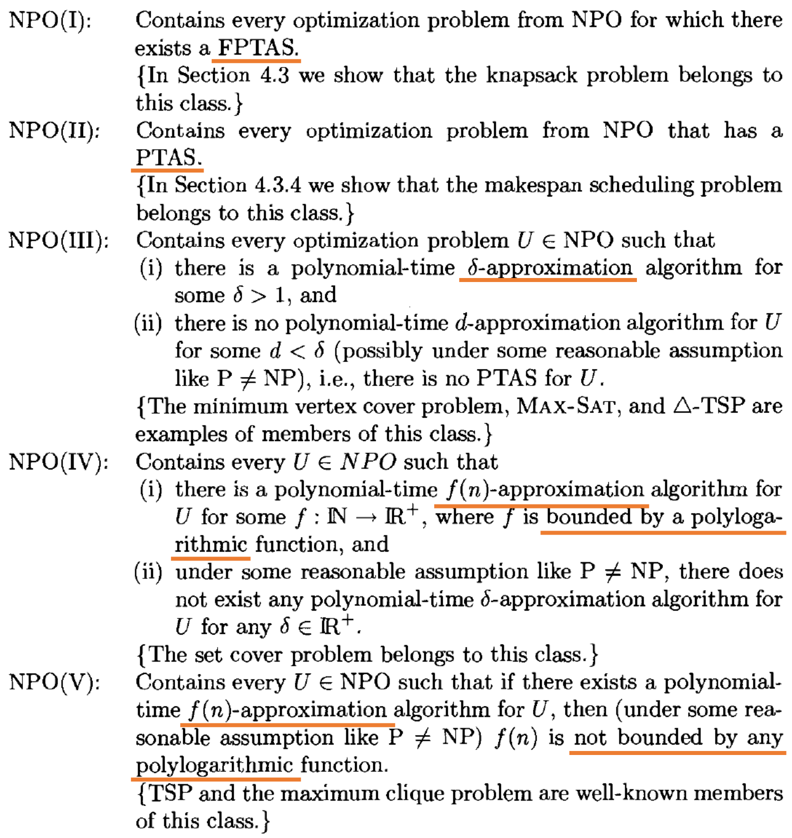
NPO(II): Contains every optimization problem from NPO that has a PTAS.
{In Section 4.3.4 we show that the makespan scheduling problem belongs to this class.}

NPO(III): Contains every optimization problem $U \in$ NPO such that
(i) there is a polynomial-time δ-approximation algorithm for some $\delta > 1$, and
(ii) there is no polynomial-time d-approximation algorithm for U for some $d < \delta$ (possibly under some reasonable assumption like P $\neq$ NP), i.e., there is no PTAS for U.
{The minimum vertex cover problem, MAX-SAT, and $\triangle$-TSP are examples of members of this class.}

NPO(IV): Contains every $U \in NPO$ such that
(i) there is a polynomial-time $f(n)$-approximation algorithm for U for some $f : \mathbb{N} \to \mathbb{R}^+$, where f is bounded by a polylogarithmic function, and
(ii) under some reasonable assumption like P $\neq$ NP, there does not exist any polynomial-time δ-approximation algorithm for U for any $\delta \in \mathbb{R}^+$.
{The set cover problem belongs to this class.}

NPO(V): Contains every $U \in$ NPO such that if there exists a polynomial-time $f(n)$-approximation algorithm for U, then (under some reasonable assumption like P $\neq$ NP) $f(n)$ is not bounded by any polylogarithmic function.
{TSP and the maximum clique problem are well-known members of this class.}

你理解这套分类体系了吗？



NPO(I): Contains every optimization problem from NPO for which there exists a FPTAS.
{In Section 4.3 we show that the knapsack problem belongs to this class.}

NPO(II): Contains every optimization problem from NPO that has a PTAS.
{In Section 4.3.4 we show that the makespan scheduling problem belongs to this class.}

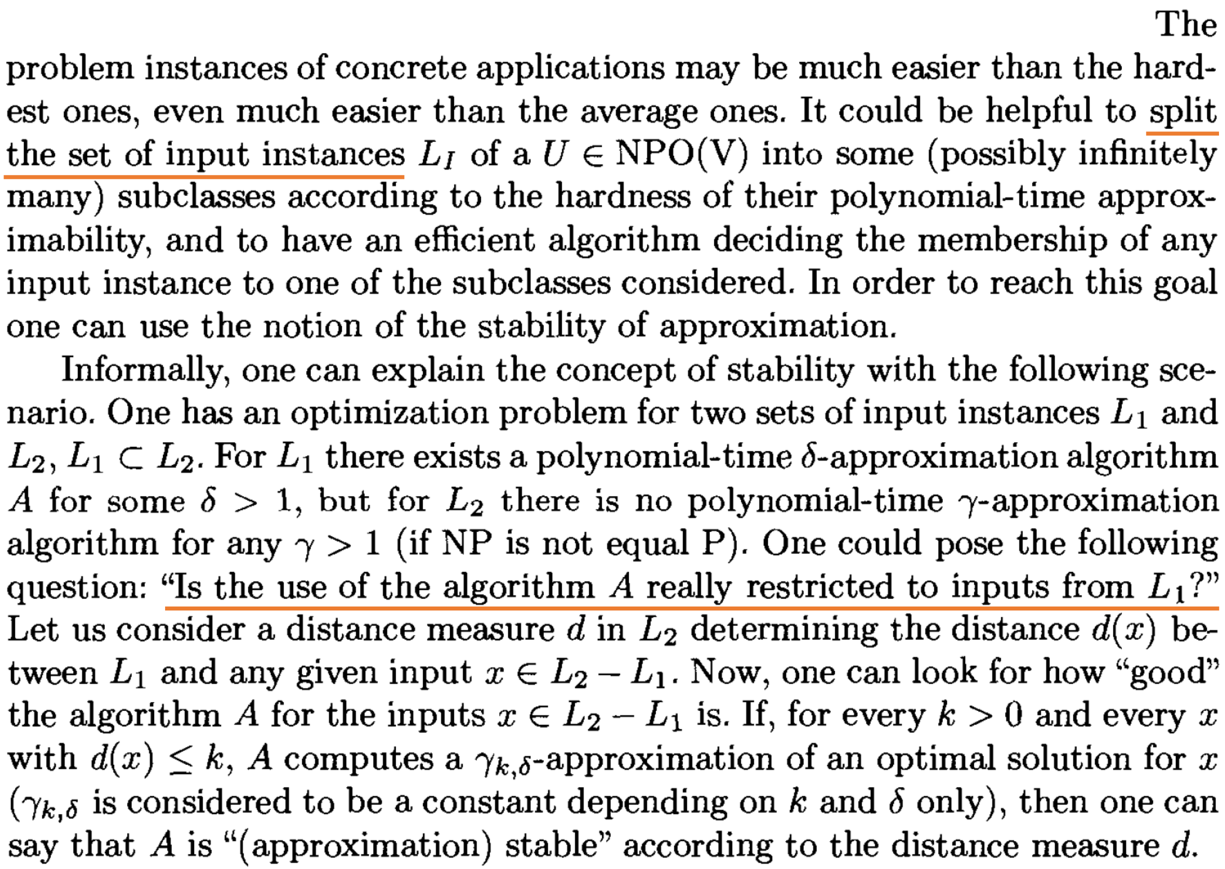
NPO(III): Contains every optimization problem $U \in$ NPO such that
(i) there is a polynomial-time δ-approximation algorithm for some $\delta > 1$, and
(ii) there is no polynomial-time d-approximation algorithm for U for some $d < \delta$ (possibly under some reasonable assumption like P $\neq$ NP), i.e., there is no PTAS for U.
{The minimum vertex cover problem, MAX-SAT, and $\triangle$-TSP are examples of members of this class.}

NPO(IV): Contains every $U \in NPO$ such that
(i) there is a polynomial-time $f(n)$-approximation algorithm for U for some $f : \mathbb{N} \to \mathbb{R}^+$, where f is bounded by a polylogarithmic function, and
(ii) under some reasonable assumption like P $\neq$ NP, there does not exist any polynomial-time δ-approximation algorithm for U for any $\delta \in \mathbb{R}^+$.
{The set cover problem belongs to this class.}

NPO(V): Contains every $U \in$ NPO such that if there exists a polynomial-time $f(n)$-approximation algorithm for U, then (under some reasonable assumption like P $\neq$ NP) $f(n)$ is not bounded by any polylogarithmic function.
{TSP and the maximum clique problem are well-known members of this class.}

你能解释stability of approximation的意义吗？

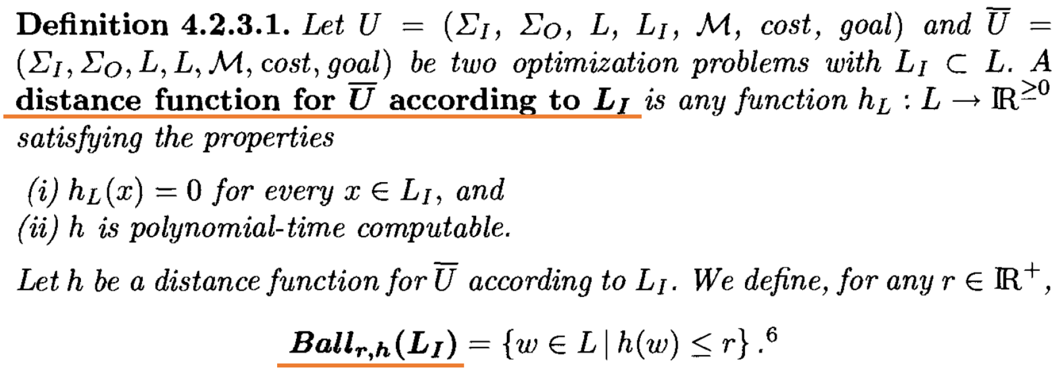
你能解释stability of approximation的意义吗？



The problem instances of concrete applications may be much easier than the hardest ones, even much easier than the average ones. It could be helpful to split the set of input instances L_I of a $U \in \mathrm{NPO(V)}$ into some (possibly infinitely many) subclasses according to the hardness of their polynomial-time approximability, and to have an efficient algorithm deciding the membership of any input instance to one of the subclasses considered. In order to reach this goal one can use the notion of the stability of approximation.

Informally, one can explain the concept of stability with the following scenario. One has an optimization problem for two sets of input instances L_1 and L_2, $L_1 \subset L_2$. For L_1 there exists a polynomial-time δ-approximation algorithm A for some $\delta > 1$, but for L_2 there is no polynomial-time γ-approximation algorithm for any $\gamma > 1$ (if NP is not equal P). One could pose the following question: "Is the use of the algorithm A really restricted to inputs from L_1?" Let us consider a distance measure d in L_2 determining the distance $d(x)$ between L_1 and any given input $x \in L_2 - L_1$. Now, one can look for how "good" the algorithm A for the inputs $x \in L_2 - L_1$ is. If, for every $k > 0$ and every x with $d(x) \leq k$, A computes a $\gamma_{k,\delta}$-approximation of an optimal solution for x ($\gamma_{k,\delta}$ is considered to be a constant depending on k and δ only), then one can say that A is "(approximation) stable" according to the distance measure d.

你能解释这些概念吗？



Definition 4.2.3.1. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *and* $\overline{U} = (\Sigma_I, \Sigma_O, L, L, \mathcal{M}, cost, goal)$ *be two optimization problems with* $L_I \subset L$. *A* **distance function for** $\overline{U}$ **according to** L_I *is any function* $h_L : L \to \mathbb{R}^{\geq 0}$ *satisfying the properties*

(i) $h_L(x) = 0$ *for every* $x \in L_I$, *and*
(ii) h *is polynomial-time computable.*

Let h *be a distance function for* $\overline{U}$ *according to* L_I. *We define, for any* $r \in \mathbb{R}^+$,

$$\boldsymbol{Ball_{r,h}(L_I)} = \{w \in L \mid h(w) \leq r\}.^6$$

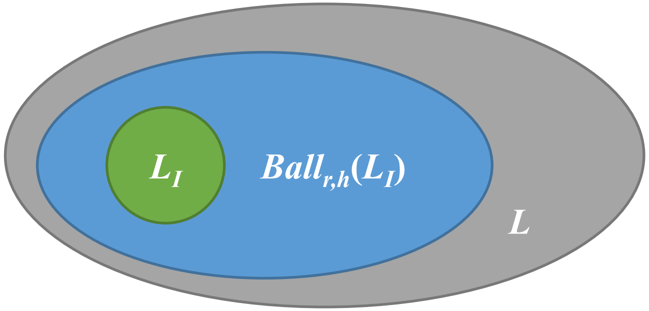
你能解释这些概念吗？

Definition 4.2.3.1. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *and* $\overline{U} = (\Sigma_I, \Sigma_O, L, L, \mathcal{M}, cost, goal)$ *be two optimization problems with* $L_I \subset L$. *A* **distance function for $\overline{U}$ according to L_I** *is any function* $h_L : L \rightarrow \mathbb{R}^{\geq 0}$ *satisfying the properties*

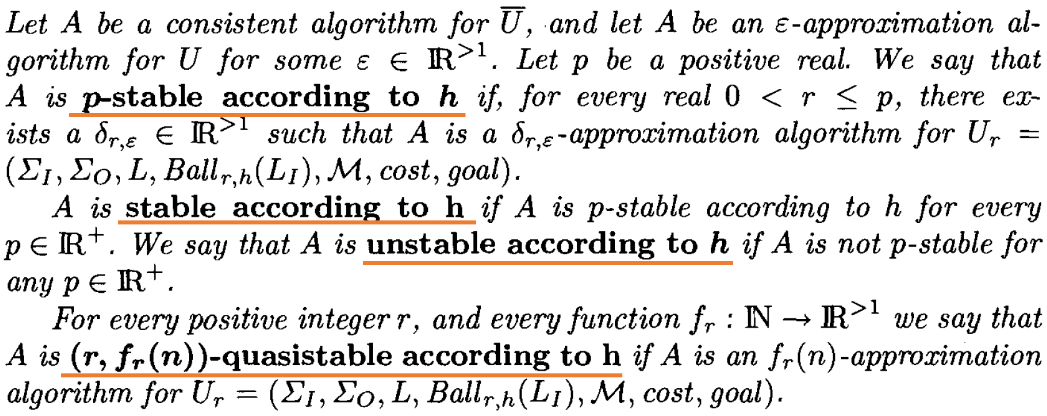
(i) $h_L(x) = 0$ *for every* $x \in L_I$, *and*
(ii) h *is polynomial-time computable.*

Let h *be a distance function for* $\overline{U}$ *according to* L_I. *We define, for any* $r \in \mathbb{R}^+$,

$$\boldsymbol{Ball_{r,h}(L_I)} = \{w \in L \mid h(w) \leq r\}.^{6}$$



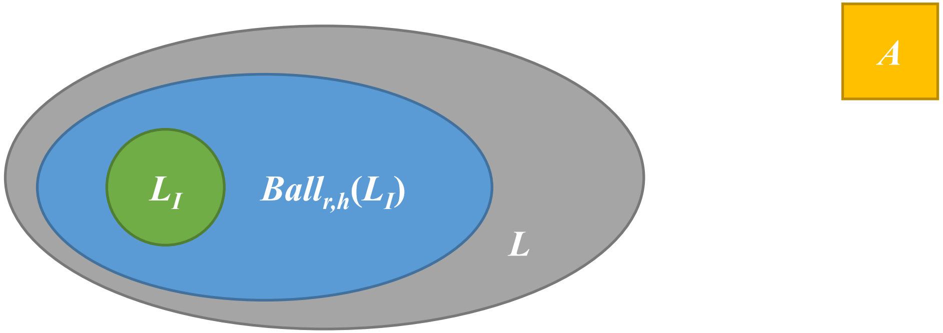
你能解释这些概念吗？



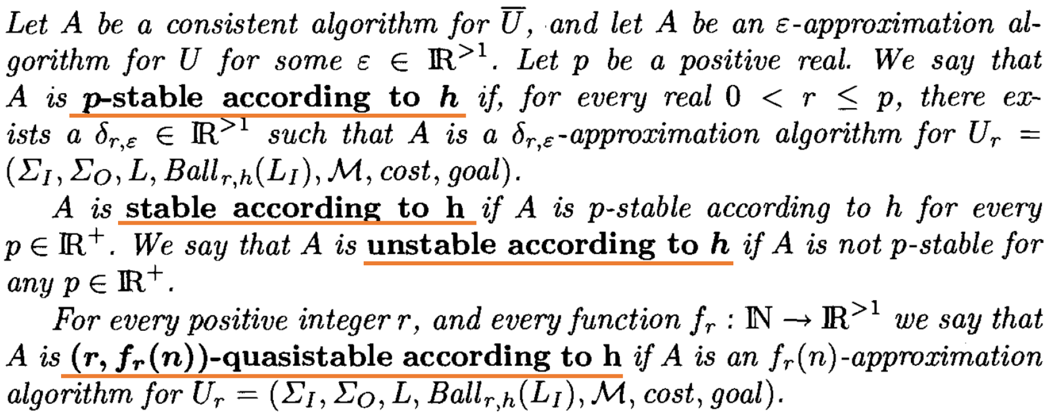
Let A be a consistent algorithm for $\overline{U}$, and let A be an ε-approximation algorithm for U for some $\varepsilon \in \mathbb{R}^{>1}$. Let p be a positive real. We say that A is **p-stable according to h** *if, for every real $0 < r \leq p$, there exists a $\delta_{r,\varepsilon} \in \mathbb{R}^{>1}$ such that A is a $\delta_{r,\varepsilon}$-approximation algorithm for $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$.*

A is **stable according to h** *if A is p-stable according to h for every $p \in \mathbb{R}^+$. We say that A is* **unstable according to h** *if A is not p-stable for any $p \in \mathbb{R}^+$.*

For every positive integer r, and every function $f_r : \mathbb{N} \rightarrow \mathbb{R}^{>1}$ we say that A is **$(r, f_r(n))$-quasistable according to h** *if A is an $f_r(n)$-approximation algorithm for $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$.*



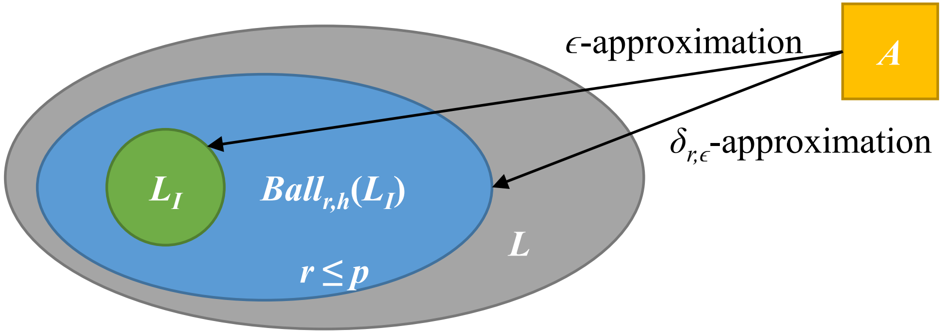
你能解释这些概念吗？



Let A be a consistent algorithm for $\overline{U}$, and let A be an ε-approximation algorithm for U for some $\varepsilon \in \mathbb{R}^{>1}$. Let p be a positive real. We say that A is **p-stable according to *h*** *if, for every real $0 < r \le p$, there exists a $\delta_{r,\varepsilon} \in \mathbb{R}^{>1}$ such that A is a $\delta_{r,\varepsilon}$-approximation algorithm for $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$.*

A is **stable according to h** *if A is p-stable according to h for every $p \in \mathbb{R}^{+}$. We say that A is* **unstable according to *h*** *if A is not p-stable for any $p \in \mathbb{R}^{+}$.*

For every positive integer r, and every function $f_r : \mathbb{N} \to \mathbb{R}^{>1}$ we say that A is **$(r, f_r(n))$-quasistable according to h** *if A is an $f_r(n)$-approximation algorithm for $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$.*



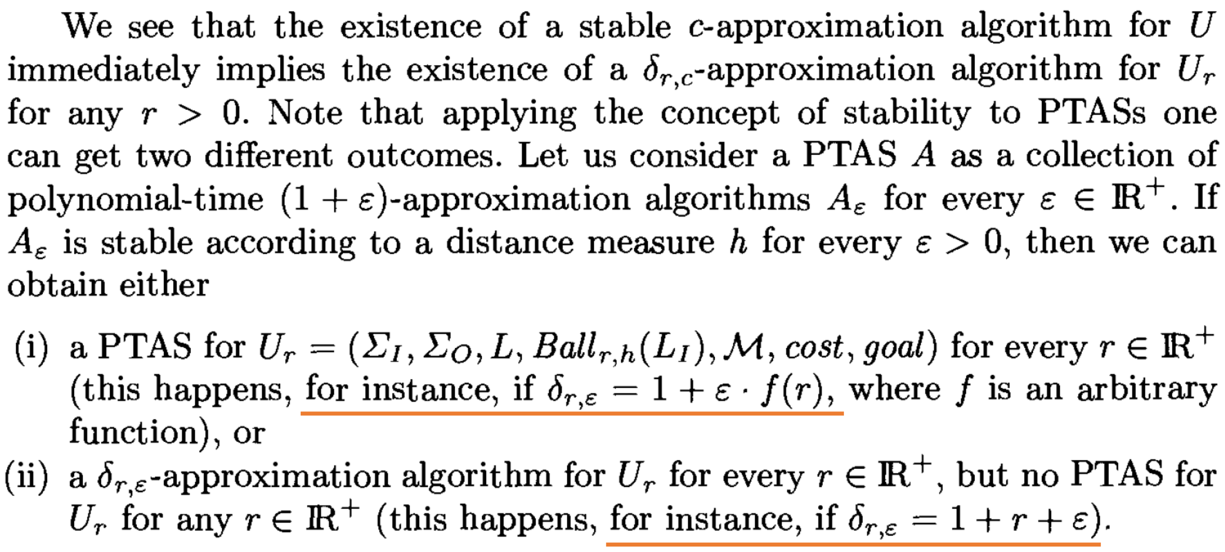
如果算法*A*是stable，那么*A*是PTAS吗？

Let A be a consistent algorithm for $\overline{U}$, and let A be an ε-approximation algorithm for U for some $\varepsilon \in \mathbb{R}^{>1}$. Let p be a positive real. We say that A is **p-stable according to h** *if, for every real $0 < r \leq p$, there exists a $\delta_{r,\varepsilon} \in \mathbb{R}^{>1}$ such that A is a $\delta_{r,\varepsilon}$-approximation algorithm for $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$.*

A is **stable according to h** *if A is p-stable according to h for every $p \in \mathbb{R}^+$. We say that A is* **unstable according to h** *if A is not p-stable for any $p \in \mathbb{R}^+$.*

Definition 4.2.1.6. *Let $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ be an optimization problem. An algorithm A is called a* **polynomial-time approximation scheme (PTAS) for U**, *if, for every input pair $(x, \varepsilon) \in L_I \times \mathbb{R}^+$, A computes a feasible solution $A(x)$ with a relative error at most ε, and $Time_A(x, \varepsilon^{-1})$ can be bounded by a function*[3] *that is polynomial in $|x|$. If $Time_A(x, \varepsilon^{-1})$ can be bounded by a function that is polynomial in both $|x|$ and ε^{-1}, then we say that A is a* **fully polynomial-time approximation scheme (FPTAS) for U**.

如果算法A是stable，那么A是PTAS吗？



We see that the existence of a stable c-approximation algorithm for U immediately implies the existence of a $\delta_{r,c}$-approximation algorithm for U_r for any $r > 0$. Note that applying the concept of stability to PTASs one can get two different outcomes. Let us consider a PTAS A as a collection of polynomial-time $(1+\varepsilon)$-approximation algorithms A_ε for every $\varepsilon \in \mathbb{R}^+$. If A_ε is stable according to a distance measure h for every $\varepsilon > 0$, then we can obtain either

(i) a PTAS for $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$ for every $r \in \mathbb{R}^+$ (this happens, for instance, if $\delta_{r,\varepsilon} = 1 + \varepsilon \cdot f(r)$, where f is an arbitrary function), or

(ii) a $\delta_{r,\varepsilon}$-approximation algorithm for U_r for every $r \in \mathbb{R}^+$, but no PTAS for U_r for any $r \in \mathbb{R}^+$ (this happens, for instance, if $\delta_{r,\varepsilon} = 1 + r + \varepsilon$).

- 什么时候是PTAS？什么时候不是？
- 你理解这两个例子了吗？

你能解释这三种distance function吗？

$$dist(G,c) = \max\left\{0, \max\left\{\frac{c(\{u,v\})}{c(\{u,p\}) + c(\{p,v\})} - 1 \,\middle|\, u,v,p \in V(G),\right.\right.$$
$$\left.\left. u \neq v, u \neq p, v \neq p\right\}\right\},$$

$$dist_k(G,c) = \max\left\{0, \max\left\{\frac{c(\{u,v\})}{\sum_{i=1}^{m} c(\{p_i,p_{i+1}\})} - 1 \,\middle|\, u,v \in V(G) \text{ and}\right.\right.$$
$$u = p_1, p_2, \ldots, p_m = v \text{ is a simple path between } u \text{ and } v$$
$$\left.\left.\text{of length at most } k \text{ (i.e., } m+1 \leq k)\right\}\right\}$$

$$distance(G,c) = \max\{dist_k(G,c) \,|\, 2 \leq k \leq |V(G)| - 1\}.$$

什么是一种好的distance function?

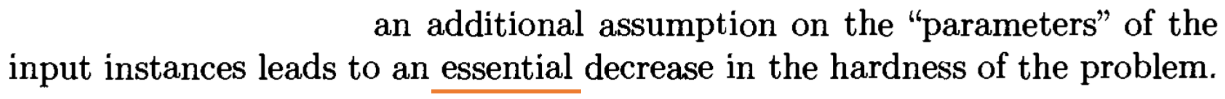
什么是一种好的distance function？

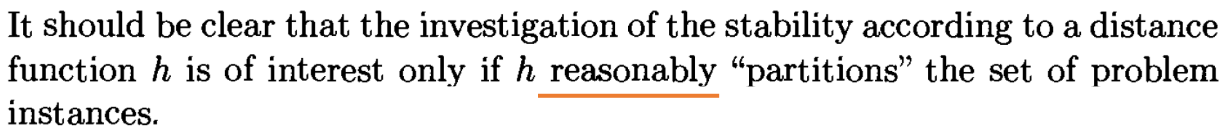
an additional assumption on the “parameters” of the input instances leads to an essential decrease in the hardness of the problem.

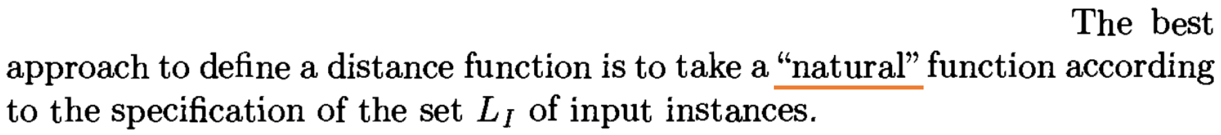
It should be clear that the investigation of the stability according to a distance function h is of interest only if h reasonably “partitions” the set of problem instances.

The best approach to define a distance function is to take a “natural” function according to the specification of the set L_I of input instances.

什么是一种好的distance function?


an additional assumption on the "parameters" of the input instances leads to an essential decrease in the hardness of the problem.


It should be clear that the investigation of the stability according to a distance function h is of interest only if h reasonably "partitions" the set of problem instances.


The best approach to define a distance function is to take a "natural" function according to the specification of the set L_I of input instances.

- 你能为某个难问题定义一种distance function吗?

OT

- 除MS和MAX-CUT外，近似算法还可用于解决其它问题，例如SCP（JH算法4.3.2.11）、SKP（JH算法4.3.4.1和4.3.4.2）等，请调研至少2种近似算法（其中至多1种来自上述例子，图上的优化问题不在调研范围内），结合例子介绍算法的设计与分析，重点阐述近似比的证明过程。