- 作业讲解
  - JH第3章练习3.6.1.3
  - 如果一个局部搜索算法给出的解一定是全局最优解，则这个算法称为“精确局部搜索算法”：(1)证明解TSP的2-exchange不是精确的。~~(2)证明解TSP的(n-1)-exchange是精确的。~~
  - 考虑带权图的最大边割集问题……
  - JH第3章练习3.7.2.1、3.7.2.4、3.7.2.5、3.7.4.4、3.7.4.12、3.7.4.16

考虑带权图的最大边割集问题: 对顶点集*V*的任一子集*S*，分割*S*和*V*-*S*的边割集的权是所有恰好一个端点在*S*中的所有边的权之和，记为*w*(*S*)，最大边割集问题就是求使得*w*(*S*)为最大的*S*. 给局部搜索算法如下：

start with any $S \subseteq V$
while there is a subset $S' \subseteq V$ such that
$||S'| - |S|| = 1$ and $w(S') > w(S)$ do:
set $S = S'$

(1) 该算法是否精确，如不是，所给的解与最优解最多差多少？

(2) 该算法是否为多项式算法?

1. Start with an arbitrary partition (for example, $\emptyset$).
2. Pick a vertex $v \in V$ such that moving it across the partition would yield a greater cut value.
3. Repeat step 2 until no such v exists.

（最坏情况为指数时间）

Proof: First of all, we observe that the maximum cut value cannot be larger than the sum of all edge weights, thus giving

$$\sum_{e \in E} w_e \geq OPT.$$

We say that an edge *contributes* to a cut if its endpoints lie in different subsets of the cut. Let S be a cut produced by our algorithm. Let v be a vertex in S. Consider the set E_v of edges incident to v. If we move v from S to $S' = V \backslash S$, edges in E_v that contributed to S become non-contributing, and vice versa. Edges not in E_v are not affected. Since S is a local optimum, moving v to S' does not increase the cut value. Therefore we have

$$\begin{aligned}
\sum_{\substack{u \in S' \\ (u,v) \in E}} w_{u,v} &\geq \sum_{\substack{u \in S \\ (u,v) \in E}} w_{u,v} \\
2 \sum_{\substack{u \in S' \\ (u,v) \in E}} w_{u,v} &\geq \sum_{\substack{u \in S \\ (u,v) \in E}} w_{u,v} + \sum_{\substack{u \in S' \\ (u,v) \in E}} w_{u,v} \\
\sum_{\substack{u \in S' \\ (u,v) \in E}} w_{u,v} &\geq \frac{1}{2} \sum_{u:(u,v) \in E} w_{u,v}.
\end{aligned}$$

Similarly, for any $v' \in S'$ we get

$$\sum_{\substack{u \in S \\ (u,v') \in E}} w_{u,v'} \geq \frac{1}{2} \sum_{u:(u,v') \in E} w_{u,v}.$$

Summing over all vertices, we obtain the desired result:

$$2c(S) = 2 \sum_{\substack{u \in S, v \in S' \\ (u,v) \in E}} w_{u,v} \geq \sum_{e \in E} w_e \geq OPT.$$

JH第3章练习3.7.2.5

- Let $x_{ij} = \begin{cases} 1 & \text{if edge}(i,j) \text{ is in tree} \\ 0 & \text{otherwise} \end{cases}$
- Let x denote the vector formed by x_{ij}'s for all $(i,j) \in E$.
- The MST found by optimal x^*, denoted T^*, will be a subgraph $T^* = (V, E^*)$, where $E^* = \{(i,j) \in E : x^*_{ij} = 1\}$ denotes the selected edge into the spanning tree.
- Subtour elimination formulation is based on the fact that T has no simple cycles and has $n-1$ edges

$$\textbf{[MST1]} \quad \min_x \sum_{(i,j)\in E} \phi_{ij} x_{ij}$$

$$s.t. \begin{cases} \sum_{(i,j)\in E} x_{ij} = n-1 \\ \sum_{(i,j)\in E(S)} x_{ij} \le |S| - 1, \ \forall S \subset V, S \neq V, S \neq \emptyset \\ x_{ij} \in \{0,1\}, \ \forall (i,j) \in E \end{cases}$$

where $E(S) \subset E$ is a subset of edges with both ends in subset $S \subset V$. Constraint $\sum_{(i,j)\in E(S)} x_{ij} \le |S| - 1$ ensures that there is no cycles in subset S.

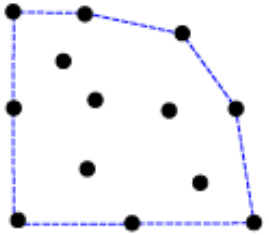
JH第3章练习3.7.4.12

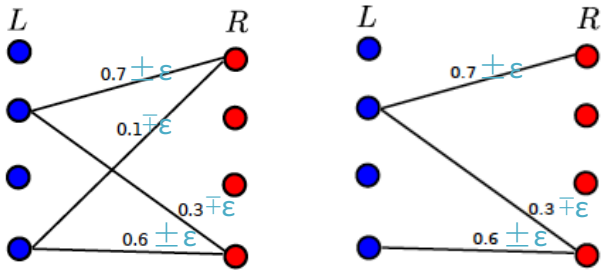
Exercise 3.7.4.12. Prove, that if G is a bipartite graph, then

(i) any optimal solution to $I(G)$ is a Boolean solution, i.e., $Opt_{\mathrm{LP}}(I(G)) = Opt_{\mathrm{MMP}}(G)$, and

(ii) any optimal solution to $Dual(I(G))$ is a Boolean solution, i.e., $Opt_{\mathrm{LP}}(Dual(I(G))) = Opt_{\mathrm{VCP}}(G)$.

- (i) 一种证明思路：
  http://www-bcf.usc.edu/~shaddin/cs599fa13/slides/lec16.pdf
  - LP的可行解构成凸包
  - LP的最优解必在凸包的顶点上
  - 非布尔解必为凸包内某线段中点→非顶点（证明有一定技巧性）





JH第3章练习3.7.4.16

- 以更一般的weighted set cover为例

IP

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j \\ \text{st} \quad & \sum_{i \in S_j} x_j \geq 1 \qquad \forall i \in \{1, ..., m\} \\ & x_j \in \{0, 1\} \qquad \forall j \in \{1, \ldots, n\} \end{aligned}$$

LP

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j \\ \text{st} \quad & \sum_{j: i \in S_j} x_j \geq 1 \qquad \forall i \in \{1, \ldots, m\} \\ & x_j \geq 0 \qquad \forall j \in \{1, \ldots, n\} \end{aligned}$$

Dual

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} y_i \\ \text{st} \quad & \sum_{i \in S_j} y_i \leq c_j \qquad \forall j \in \{1, \ldots, n\} \\ & y_i \geq 0 \qquad \forall i \in \{1, \ldots, m\} \end{aligned}$$

Conditions

$For\ each\ 1 \leq j \leq n:\ either\ x_j = 0\ or \sum_{i \in S_j} y_i = c_j$ （称作tight）

$For\ each\ 1 \leq i \leq m:\ either\ y_i = 0\ or \sum_{j: i \in S_j} x_j \leq$ k （该条件恒成立，不用管）

1: Initialize $x = 0$ and $y = 0$
2: **while** $U \neq \emptyset$ **do**
3: Choose an uncovered element, say i, and raise y_i until some set in $\mathcal{S}$ goes tight, say S_j
4: Choose all these sets S_j and set $x_j = 1$
5: Remove all the elements in these sets S_j from U
6: Remove all these sets S_j from the collections $\mathcal{S}$
7: **end while**
8: Return $\mathcal{C} = \{S_j \mid x_j = 1\}$

- 教材讨论
  - JH第4章第1、2节
  - JH第4章第3节第1、2、3小节

问题1：近似算法的基本概念

- 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.

- 你理解这些概念了吗？
  - relative error $\varepsilon_A(x) = \frac{|cost(A(x)) - Opt_U(x)|}{Opt_U(x)}$.

    $\varepsilon_A(n) = \max\{\varepsilon_A(x) \mid x \in L_I \cap (\Sigma_I)^n\}$.
  - approximation ratio

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

    $R_A(n) = \max\{R_A(x) \mid x \in L_I \cap (\Sigma_I)^n\}$.
  - δ-approximation algorithm $R_A(x) \leq \delta$ *for every* $x \in L_I$.
  - f(n)-approximation algorithm $R_A(n) \leq f(n)$ *for every* $n \in \mathbb{N}$.
- 你理解PTAS和FPTAS了吗？它们的区别是什么？

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.

问题2：MIN-VCP

- 这个算法的基本思路是什么？
- 近似比是多少，为什么？
- 时间复杂度是多少？为什么？用什么数据结构来实现？

Algorithm 4.3.2.1. Input: A graph $G = (V, E)$.

Step 1: $C := \emptyset$ {during the computation $C \subseteq V$, and at the end C should contain a vertex cover};
$A := \emptyset$ {during the computation $A \subseteq E$ is a matching, and at the end A is a maximal matching};
$E' := E$ {during the computation $E' \subseteq E$, E' contains exactly the edges that are not covered by the actual C, and at the end $E' = \emptyset$}.

Step 2: **while** $E' \neq \emptyset$
do begin choose an arbitrary edge $\{u, v\}$ from E';
$C := C \cup \{u, v\}$;
$A := A \cup \{\{u, v\}\}$;
$E' := E' - \{$all edges incident to u or $v\}$
end

Output: C.

问题2： MIN-VCP (续)

- 你能构造出一些近似比为1的例子吗？
- 你能构造出一些近似比为2的例子吗？

Algorithm 4.3.2.1. Input: A graph $G = (V, E)$.

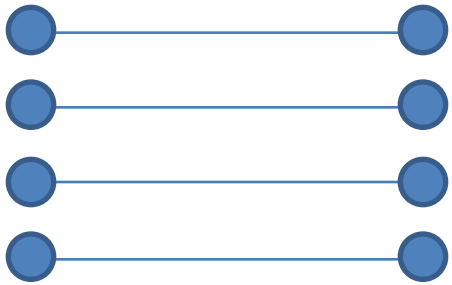
Step 1: $C := \emptyset$ {during the computation $C \subseteq V$, and at the end C should contain a vertex cover};
$A := \emptyset$ {during the computation $A \subseteq E$ is a matching, and at the end A is a maximal matching};
$E' := E$ {during the computation $E' \subseteq E$, E' contains exactly the edges that are not covered by the actual C, and at the end $E' = \emptyset$}.

Step 2: **while** $E' \neq \emptyset$
do begin choose an arbitrary edge $\{u, v\}$ from E';
$C := C \cup \{u, v\}$;
$A := A \cup \{\{u, v\}\}$;
$E' := E' - \{$all edges incident to u or $v\}$
end

Output: C.

问题2： MIN-VCP (续)

- 你能构造出一些近似比为1的例子吗？
- 你能构造出一些近似比为2的例子吗？



问题3：SCP

- 这个算法的基本思路是什么？
- 近似比是多少？
- 时间复杂度是多少？为什么？用什么数据结构来实现？

Algorithm 4.3.2.11.

Input: $(X, \mathcal{F})$, where X is a finite set, $\mathcal{F} \subseteq \mathcal{P}ot(X)$ such that $X = \bigcup_{Q \in \mathcal{F}} Q$.

Step 1: $C := \emptyset$ {during the computation $C \subseteq \mathcal{F}$ and at the end C is a set cover of $(X, \mathcal{F})$};
$U := X$ {during the computation $U \subseteq X$, $U = X - \bigcup_{Q \in C} Q$ for the actual C, and at the end $U = \emptyset$}.

Step 2: **while** $U \neq \emptyset$
do begin choose an $S \in \mathcal{F}$ such that $|S \cap U|$ is maximal;
$U := U - S$;
$C := C \cup \{S\}$
end

Output: C.

问题3：SCP (续)

- 你能构造出一些近似比为1的例子吗？
- 你能构造出一些近似比为Ω(ln*n*)的例子吗？

Algorithm 4.3.2.11.

Input: $(X, \mathcal{F})$, where X is a finite set, $\mathcal{F} \subseteq \mathcal{P}ot(X)$ such that $X = \bigcup_{Q \in \mathcal{F}} Q$.

Step 1: $C := \emptyset$ {during the computation $C \subseteq \mathcal{F}$ and at the end C is a set cover of $(X, \mathcal{F})$};

$U := X$ {during the computation $U \subseteq X$, $U = X - \bigcup_{Q \in C} Q$ for the actual C, and at the end $U = \emptyset$}.

Step 2: **while** $U \neq \emptyset$

do begin choose an $S \in \mathcal{F}$ such that $|S \cap U|$ is maximal;

$U := U - S$;

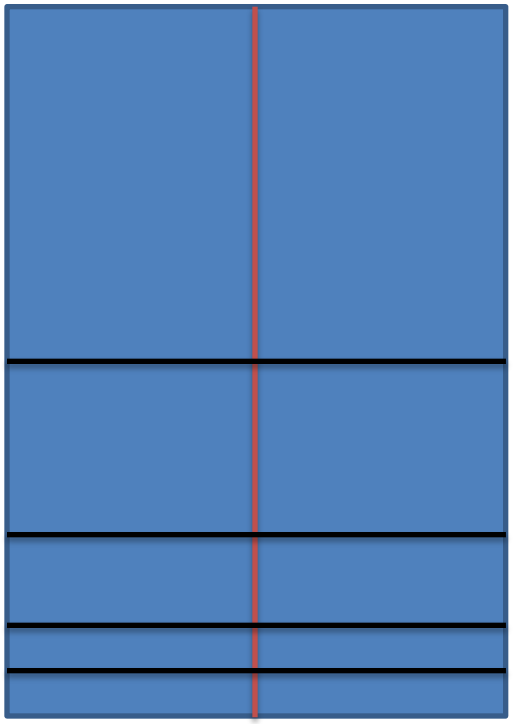
$C := C \cup \{S\}$

end

Output: C.

问题3：SCP (续)

- 你能构造出一些近似比为1的例子吗？
- 你能构造出一些近似比为Ω(ln*n*)的例子吗？
  - $U = \{ (x,y) \mid 1 \le x,y \le n \}$
  - $S_1 = \{ (x,y) \mid x \le n/2 \}$
  - $S_2 = \{ (x,y) \mid x > n/2 \}$
  - $T_1 = \{ (x,y) \mid n/2 < y \le n \}$
  - $T_2 = \{ (x,y) \mid n/4 < y \le n/2 \}$
  - $T_3 = \{ (x,y) \mid n/8 < y \le n/4 \}$
  - ……



问题3：SCP (续)

- 和MIN-VCP相比，SCP到底难在什么地方？

问题4：MAX-CUT

- 这个算法的基本思路是什么？
- 近似比是多少，为什么？
- 时间复杂度是多少？为什么？用什么数据结构来实现？

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: $(S, V - S)$.

问题4： MAX-CUT (续)

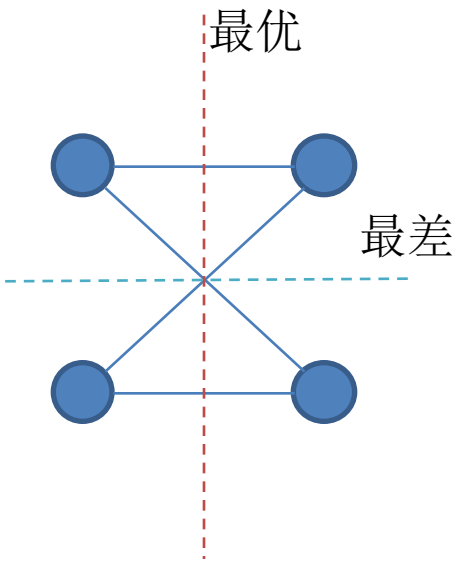
- 你能构造出一些近似比为1的例子吗？
- 你能构造出一些近似比为2的例子吗？

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: $(S, V - S)$.

问题4： MAX-CUT (续)

- 你能构造出一些近似比为1的例子吗？
- 你能构造出一些近似比为2的例子吗？



问题5：greedy和local search

- 任选1-2个问题，分别给出一种greedy算法和一种local search算法，并尽力给出近似比（或者给出一些坏例子）
  - longest simple path
  - MAX-SAT
  - MAX-CL