Metaheuristic Optimization for P-Center Problem

吕志鹏

教授，博士生导师

人工智能与优化研究所 所长

华中科技大学 计算机学院

问题背景

- P-center问题，在给定N个节点的网络中选择P个节点作为服务设施。
- P中心设备选址问题是著名的离散选址问题，属于NP-hard，其一般具有多约束、大规模、多目标和不确定性等特点。
- P-center问题有着广泛的现实应用场景，如消防站选址、物流运输网络、服务器网络等。
- 物流对于企业来说是第三利润源泉，因此为物流配送中心选好位址可以提高企业整体效益，节约企业成本，是企业发展的新战略。

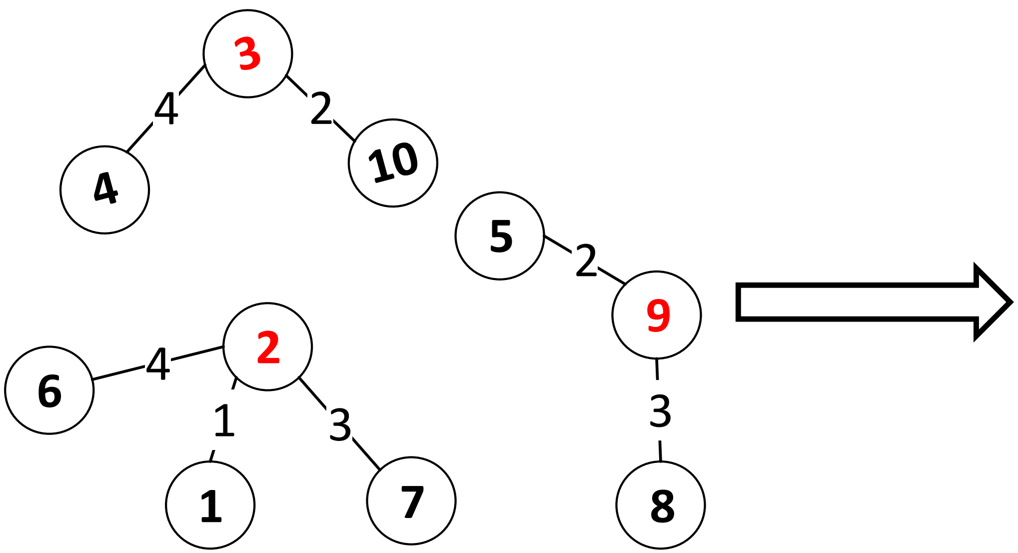
问题描述

➢**已知：**无向连通图$\boldsymbol{G(V,E)}$，图中任意两点的距离，服务点数**P**

➢**决策变量：**从**N**个结点中选择**P**个结点作为服务结点（记为集合**R**），服务剩下的**(N-P)**个用户结点（记为集合**D**），$\boldsymbol{R} \cup \boldsymbol{D} = \{\boldsymbol{V}\}$

➢ **约束条件：**每个节点由离它最近的服务点提供服务，这条边称为服务边

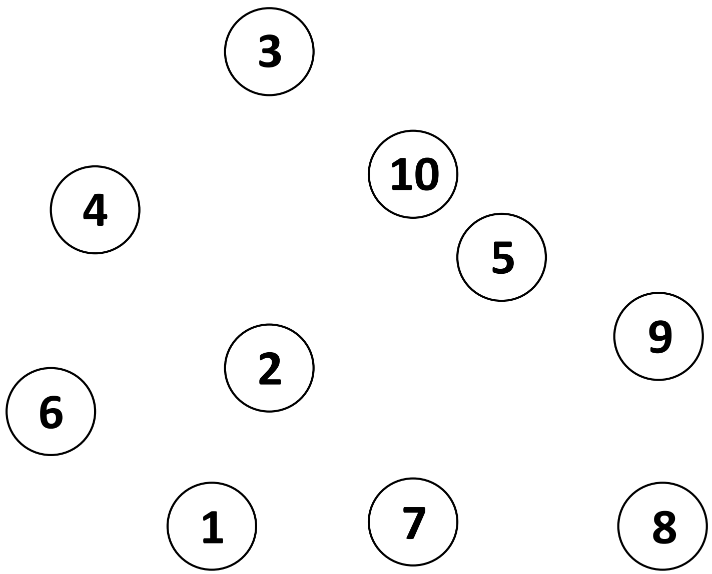
➢ **目标函数f：**所有用户结点服务边的最大值，即$f = max\{min\{d_{i,j}\}\}$



$$f = \max\{4,1,3,2,3,4,2\} \\ = 4$$

初始解

- 通过逐一选择节点作为服务节点的方式构造初始解
- 首先随意产生一个节点，如$\boldsymbol{p_1}$=2，选择最长的服务边，在最长边的用户节点w的$\boldsymbol{N_{wk}}$中前k个点中随机选择一个，即得到第二个服务节点，如$\boldsymbol{p_2}=9$ 依次类推直至找到p个服务节点
- N_{wk}：与结点w相邻的前k小节点

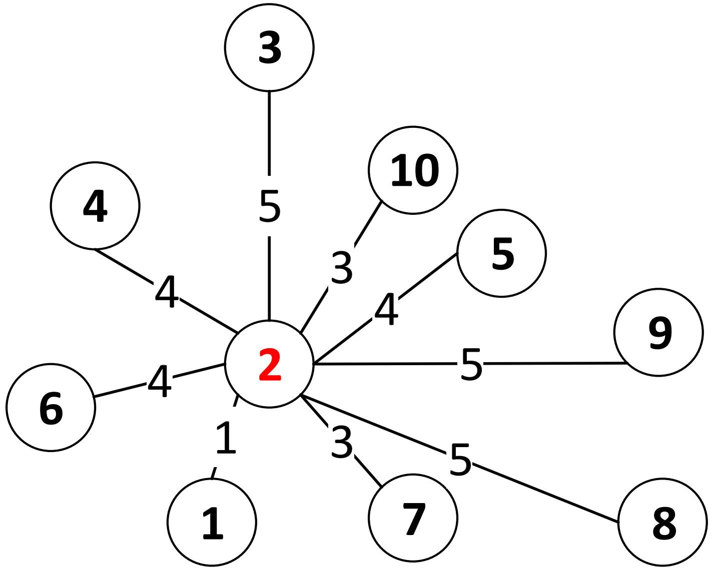


$N_9 = \{9,5,8,2,7,10,1,3,4,6\}$

此处 $k = 3$，则
$N_{93} = \{9,5,8\}$

初始解

- 通过逐一选择节点作为服务节点的方式构造初始解
- 首先随意产生一个节点，如$\boldsymbol{p_1}$=2，选择最长的服务边，在最长边的用户节点w的$\boldsymbol{N_{wk}}$ 中前k个点中随机选择一个，即得到第二个服务节点，如 $\boldsymbol{p_2}=9$ 依次类推直至找到p个服务节点
- N_{wk}：与结点w相邻的前k小节点



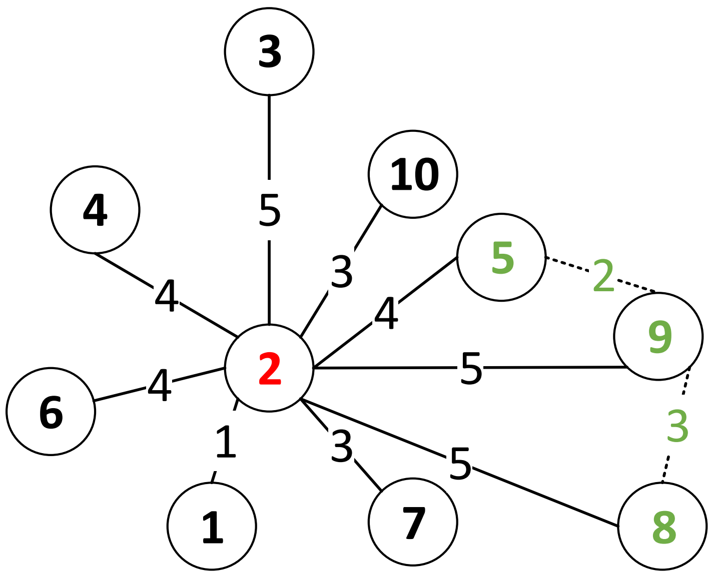
$N_9 = \{9,5,8,2,7,10,1,3,4,6\}$

此处 $k = 3$，则

$N_{93} = \{9,5,8\}$

初始解

- 通过逐一选择节点作为服务节点的方式构造初始解
- 首先随意产生一个节点，如$\boldsymbol{p_1}$=2，选择最长的服务边，在最长边的用户节点w的$\boldsymbol{N_{wk}}$ 中前k个点中随机选择一个，即得到第二个服务节点，如 $\boldsymbol{p_2}=9$ 依次类推直至找到p个服务节点
- N_{wk}：与结点w相邻的前k小节点



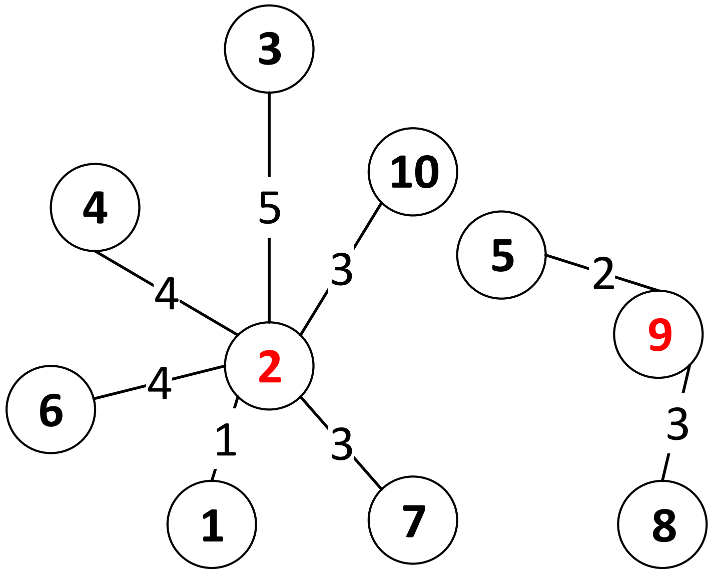
$N_9 = \{9,5,8,2,7,10,1,3,4,6\}$

此处 $k=3$，则

$N_{93} = \{9,5,8\}$

初始解

- 通过逐一选择节点作为服务节点的方式构造初始解
- 首先随意产生一个节点，如$\boldsymbol{p_1}$=2，选择最长的服务边，在最长边的用户节点w的$\boldsymbol{N_{wk}}$ 中前k个点中随机选择一个，即得到第二个服务节点，如 $\boldsymbol{p_2}=9$ 依次类推直至找到p个服务节点
- N_{wk}：与结点w相邻的前k小节点



$N_9 = \{9,5,8,2,7,10,1,3,4,6\}$

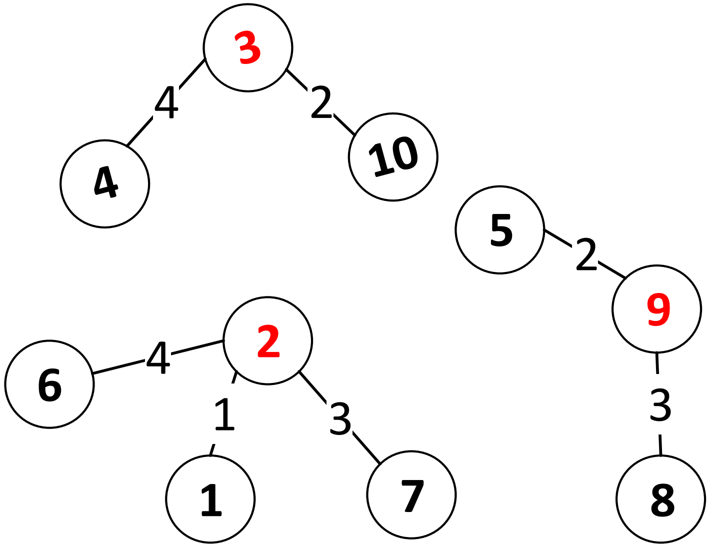
此处 $k = 3$，则

$N_{93} = \{9,5,8\}$

初始解

初始解如图：

$$f = \max\{4,1,3,2,3,4,2\}$$
$$= 4$$

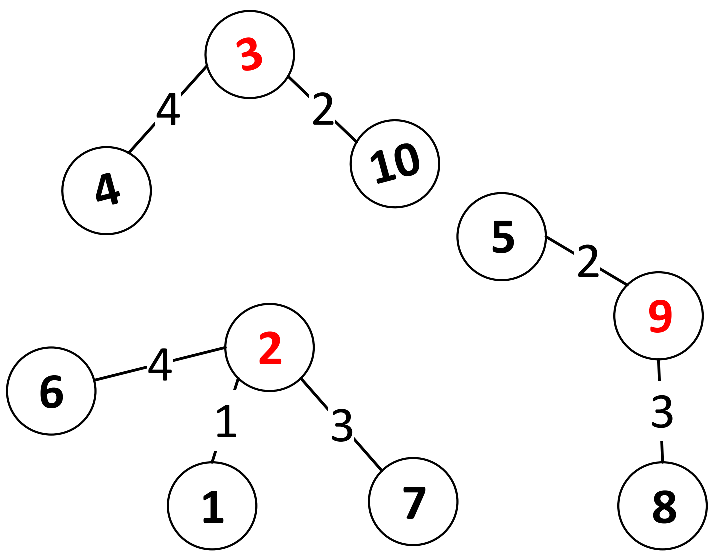


邻域结构

- 交换（Swap）：
  - 将一个用户节点和一个服务节点进行交换，则邻域大小为P*(N-P)
  - 这里只考虑“关键”的节点对的交换，也就是只考虑对应最长服务边的节点对的交换
  - 节点对的交换可以分解为“增加一个用户节点作为服务节点”+“删除一个已有的服务点”两步来完成

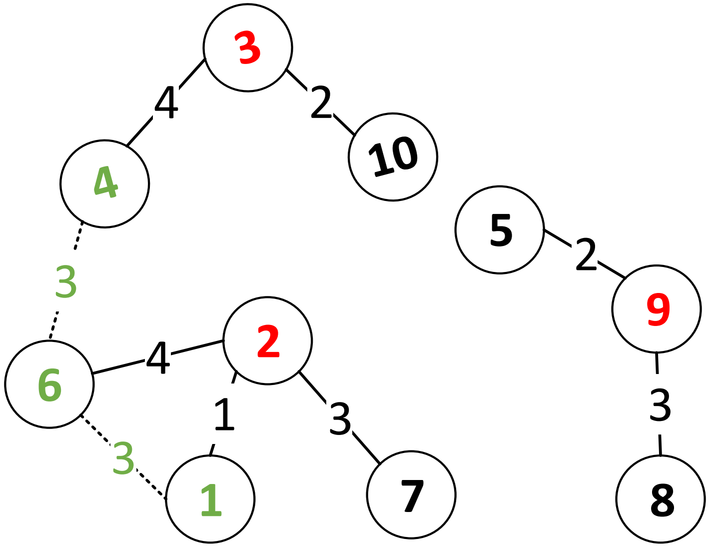
交换

- 找到最长的服务边，(2,6) 和$(3,4)$, 随机选择一条，如 (2,6)
- 从$\boldsymbol{N_{6k}}$中找出距离节点6比$\boldsymbol{d(2,6)}$小的节点集合，如 $\mathbf{N_{6k}} = \{\mathbf{6}, \mathbf{1}, \mathbf{4}\}$
- 分别试探加入 6, 1, 4 中的一个节点，变为P+1个服务点的中间解



交换

- 找到最长的服务边，(2,6) 和 $(3,4)$，随机选择一条，如 (2,6)
- 从N_{6k}中找出距离节点6比$d(2,6)$小的节点集合，如 $\mathbf{N_{6k}} = \{6, 1, 4\}$
- 分别试探加入 6, 1, 4 中的一个节点，变为P+1个服务点的中间解



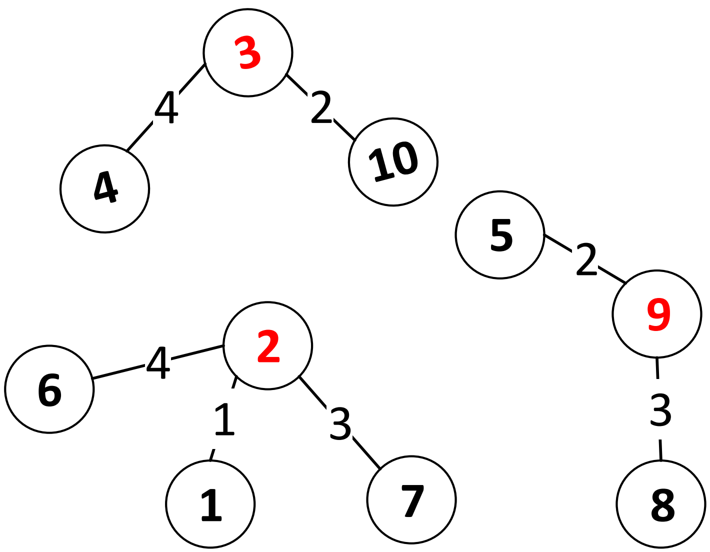
F和D表

F表： 表示该结点的最近服务结点F_i^0和次近服务结点F_i^1

D表： 表示该结点的最近服务结点的距离D_i^0和次近服务结点的距离D_i^1

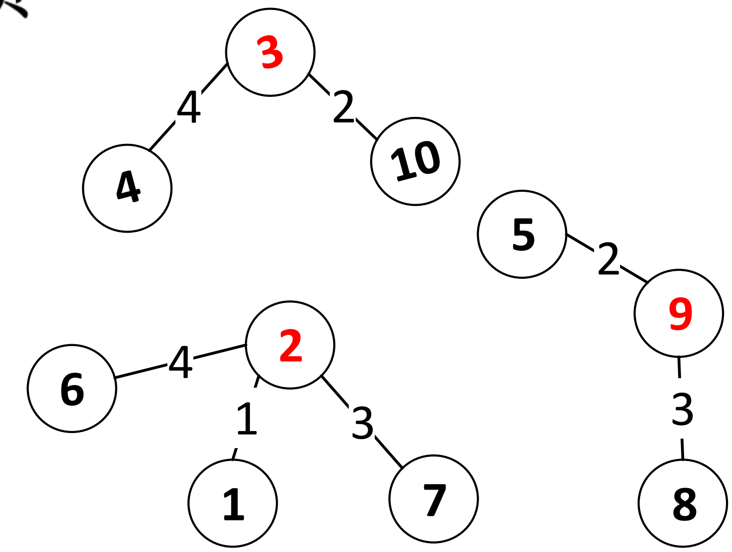
F	0（最近）	1（次近）
1	2	3
2	2	3
3	3	2
4	3	2
5	9	2
6	2	3
7	2	9
8	9	2
9	9	2
10	3	2

D	0（最近）	1（次近）
1	1	7
2	0	5
3	0	5
4	4	4
5	2	4
6	4	6
7	3	5
8	3	5
9	0	5
10	2	3



增加节点后D表和F表的更新

- 如对于用户结点6，$F_6^0=2, F_6^1=3; D_6^0=4, D_6^1=6$，若增加1为服务结点，$d(6,1)=3$。此时更新F表和D表，$d<D_6^0$，则$D_6^0=d, D_6^1=D_6^0$；同时$F_6^0=1, F_6^1=2$
- 对于每个结点都做上述处理（包括设备结点）
- 伪代码见下一页



Add_Facility（）

Procedure Add_Facility(f)

$S_c \leftarrow 0$

$S \leftarrow S \cup f$

$\forall\, v \in V$

  if $d_{fv} < D_v^0$

    $D_v^1 \leftarrow D_v^0;\ F_v^1 \leftarrow F_v^0;\ D_v^0 \leftarrow d_{fv};\ F_v^0 \leftarrow f$

  else if $d_{fv} < D_v^1$

    $D_v^1 \leftarrow d_{fv};\ F_v^1 \leftarrow f$

  if $D_v^0 > S_c$

    $S_c \leftarrow D_v^0$

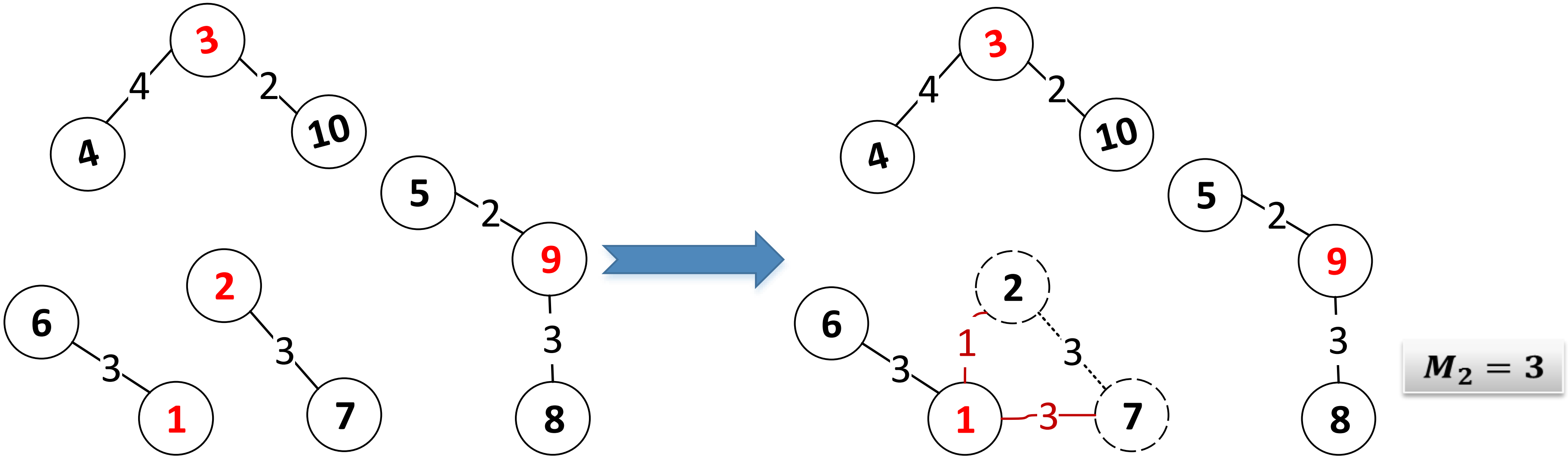
时间复杂度$O(n)$

最优交换对的选择

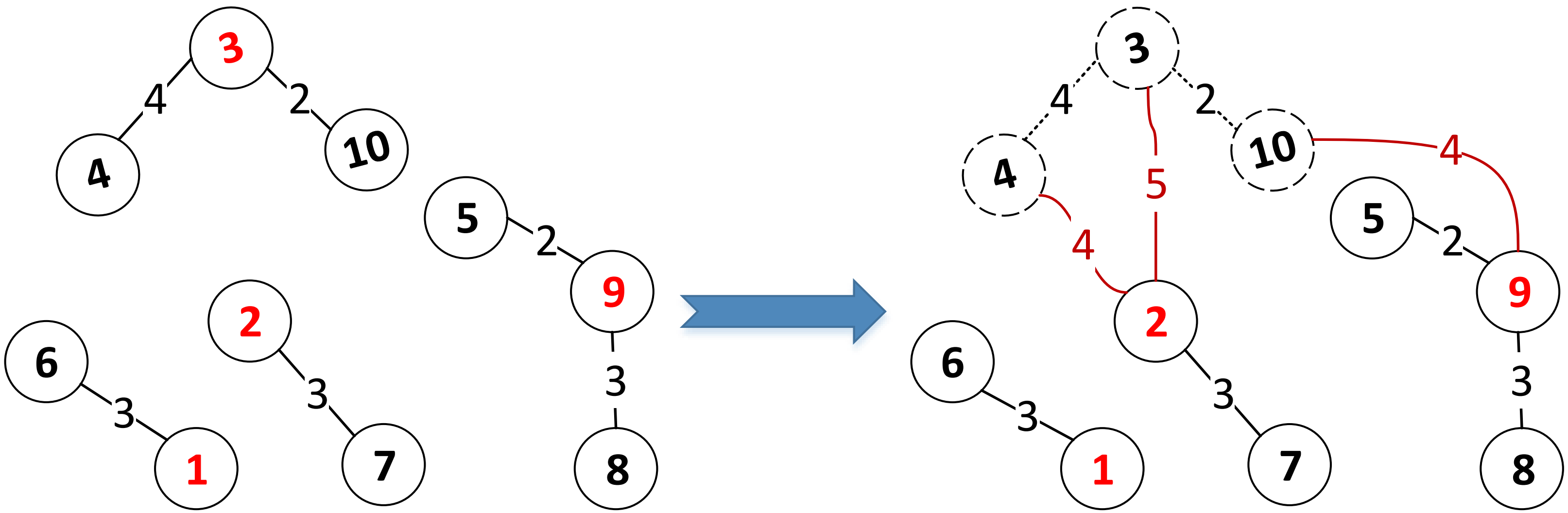
- 添加完成后得到p+1个节点的图，需要依次试探删除结点 2、3、9, 以寻求最优的交换对

- $\boldsymbol{M_f}$： 删除设备结点 $\boldsymbol{f}$ 后产生的最长服务边的距离

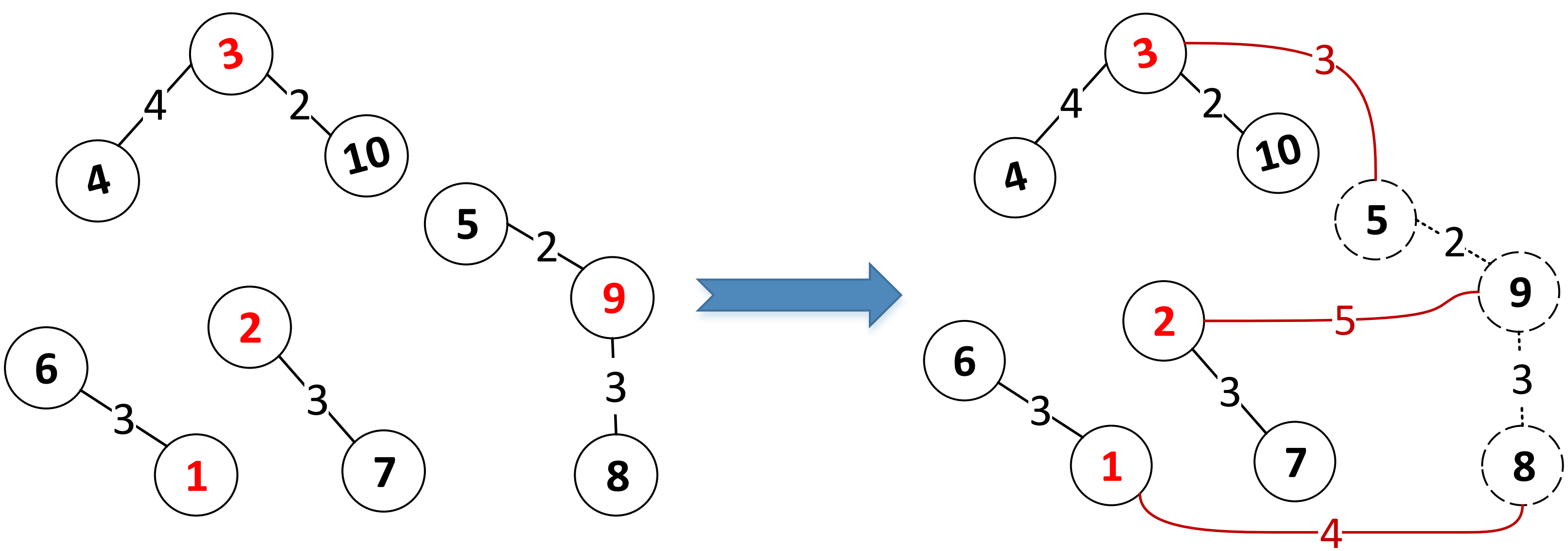
- 如删除2，则红色边为新增服务边，查D表可知 $M_2 = 3$



最优交换对的选择(续)

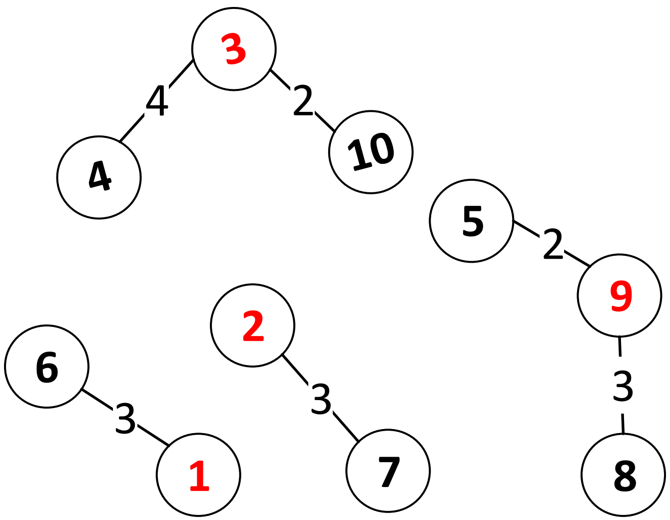


$M_3 = 5$



$M_9 = 5$

利用F表和D表计算M_f



F	0（最近）	1（次近）
1	1	2
2	2	1
3	3	2
4	3	2
5	9	2
6	1	2
7	2	1
8	9	2
9	9	2
10	3	2

D	0（最近）	1（次近）
1	0	1
2	0	1
3	0	5
4	4	4
5	2	4
6	3	4
7	3	3
8	3	5
9	0	5
10	2	3

$M_3 = 5$
$M_2 = 3$
$M_9 = 5$

删除设备时更新D表和F表

Procedure Remove_Facility(f)

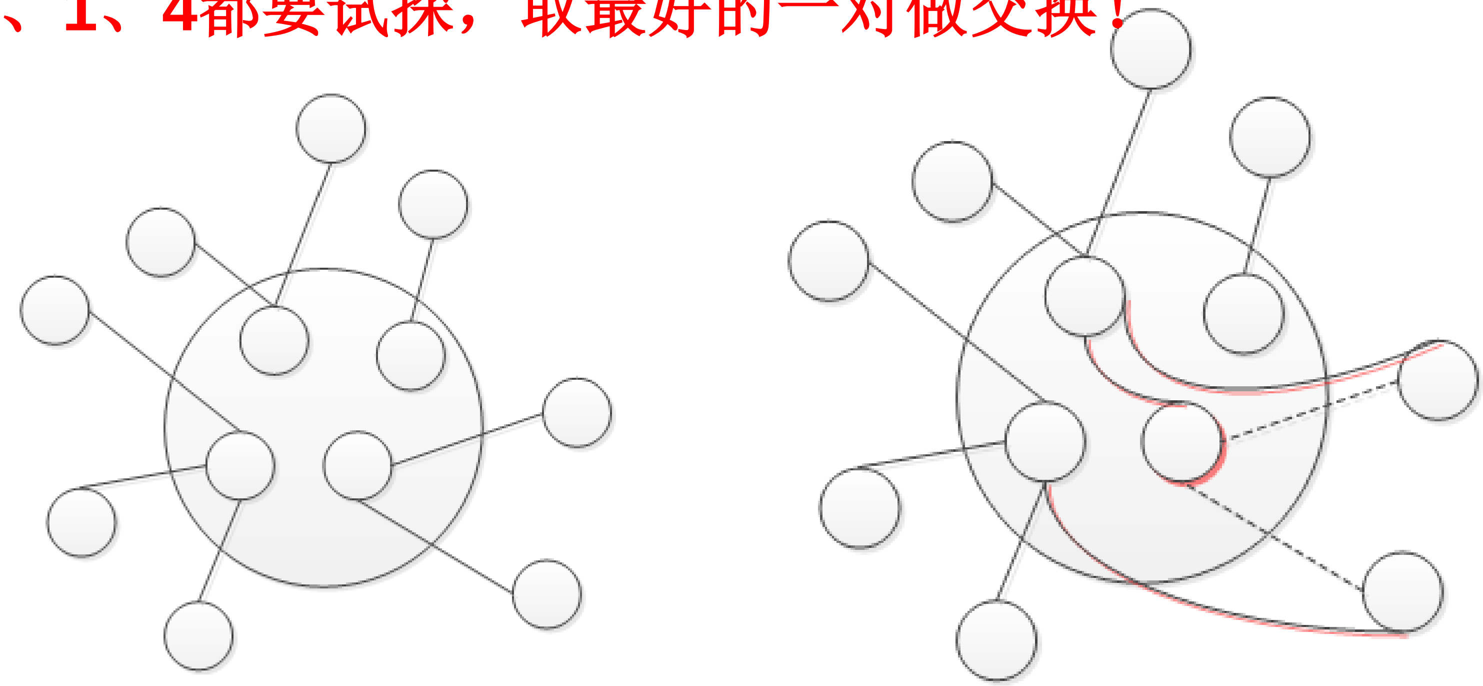
1. $S_c \leftarrow 0$
2. $S \leftarrow S - f$
3. $\forall\, v \in V$
4. if $F_v^0 = f$
5. $D_v^0 \leftarrow D_v^1$; $F_v^0 \leftarrow F_v^1$; $(D_v^1, F_v^1) \leftarrow$ Find_Next(v)
6. else if $F_v^1 = f$
7. $(D_v^1, F_v^1) \leftarrow$ Find_Next(v)
8. if $D_v^0 > S_c$
9. $S_c \leftarrow D_v^0$

最坏情况下时间复杂度$O(np)$

交换对的目标函数值

- 记录原始目标函数值 S_C
- 比较目标函数值S_C与M_f（f为被删除的结点）的大小，取其大者为新的目标函数值

结点6、1、4都要试探，取最好的一对做交换！



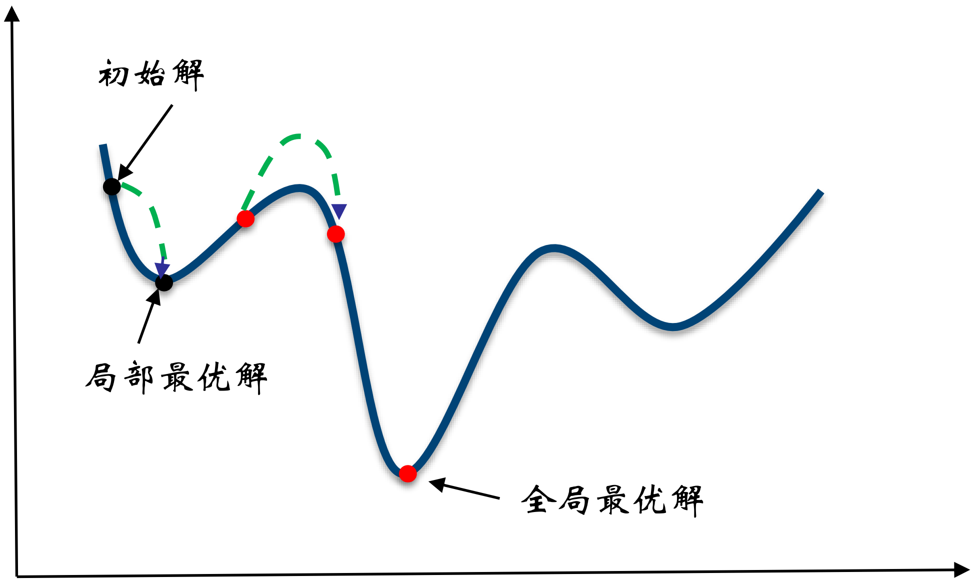
Find_Pair（）

Function Find_Pair(w)

$C \leftarrow \max(d_{ij})$
$\forall\ i \in N_{wk}$
  Add_Facility(i)
  $\forall\ f \in S, M_f \leftarrow 0$
  $\forall\ v \in V - S$
    if $\min(d_{iv}, D_v^1) > M_{F_v^0}$
      $M_{F_v^0} \leftarrow \min(d_{iv}, D_v^1)$
  $\forall\ f \in S$
    if $M_f = C$
      $L \leftarrow L \cup (f, i)$
    else if $M_f < C$
      $L \leftarrow (f, i); C \leftarrow M_f$
  Remove_Facility(i)
return $(f, i) \in L$

最坏情况下时间复杂度$\boldsymbol{O(n^2)}$
随着时间的推移，**k**值越来越小，时间复杂度降为$\boldsymbol{O(n)}$

禁忌局部搜索



- 显著的特征：

A. 能通过接受差解邻居，使搜索逃离局部最优解的陷阱；（最大的减小/最小的增大）

B. 禁忌最近执行过的动作，使得搜索不在某个小范围区域内兜圈子

Tabu Search
Escaping from Local Optimum

* Tabu Search incorporates a tabu list as a “recency-based” memory structure to assure that solutions visited within a certain span of iterations, called tabu tenure, will not be revisited.

* TS then restricts consideration to moves not forbidden by the tabu list, and selects a move that produces the best move value to perform.

Tabu Search

- 1. What?
- 2. Tenure?
- 3. How to judge if a move is forbidden?

Attributes or Solution?

- It should be noted that we generally forbid **attributes** of solutions, but not the **solutions** themselves, since it is too expensive to forbid **solutions**.

- For the tabu list, once move *<i, j>* is performed, vertex pair <i,j> is forbidden to swap again for the next *tt* iterations.

- Another alternative tabu list can be forbid i and j separately, with different tabu tenure.

Tabu Tenure

- For the tabu list, once move <*i*, *j*> is performed, vertex pair <*i*, *j*> *is* forbidden to swap for the next *tt* iterations.

- Here, the tabu tenure *tt* is dynamically determined by

$$tt = R + r(10)$$

where $r(10)$ takes a random number in {1,…,10}, say R=10.

Double Tabu List

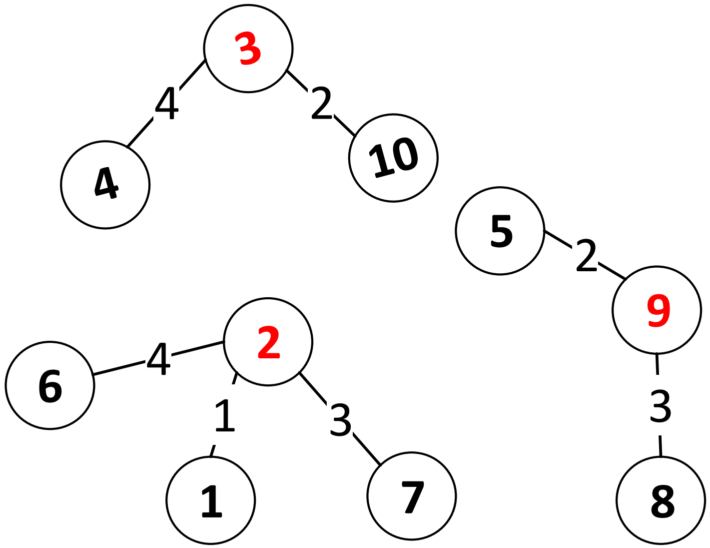
* For the tabu list, once move <*i*, *j*> is performed （where I is customer vertex and j is service vertex），vertex *I is* forbidden to become customer vertex again during the next α iterations while vertex j is forbidden to become service vertex again during the next β iterations.

$$\alpha = iter + \frac{p}{10} + rand\%10$$

$$\beta = iter + \frac{n-p}{10} + rand\%100$$

TabuTenure Table

pair	1	2	3	4	5	6
1	-	0	9	0	0	0
2	-	-	0	0	10	0
3	-	-	-	0	0	0
4	-	-	-	-	0	0
5	-	-	-	-	-	0
6	-	-	-	-	-	-



At the begining of the search, the TabuTenure table is initialized to be zero.

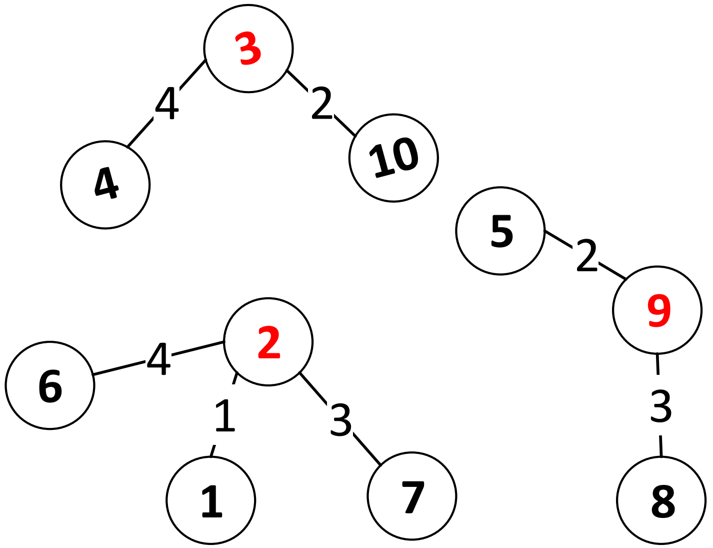
once move <*i, j*> is performed, the value of Table[i][j]=TabuTenure.

Once the search progresses, the non-zero value of the table is decreased by one at each time.

In the following search, we can decide if a move <*i, j*> is tabu by checking if Table[i][j]>0.

Fast Implementation

pair	1	2	3	4	5	6
1	-	100 +10	0	0	0	0
2	-	-	0	0	0	102 +14
3	-	-	-	101 +15	0	0
4	-	-	-	-	0	0
5	-	-	-	-	-	0
6	-	-	-	-	-	-



The above Table[i][j] records the relative tabu tenure length. Why not record the absolute tabu tenure length?

Once move <i, j> is performed, the value of Table[i][j] = TabuTenure+Iter.

In the following search, we can decide if a move <i, j> is tabu by checking if Table[i][j]>Iter.

In this way, the tabu tenure table can be updated in O(1).

Aspiration

- If one move can override the best found solution found so far, it is accepted even if it is in tabu status.

- This is because only the **attributes** but not **solutions** themselves are stored in the tabu table.

TS Algorithm

1. Generate initial solution S, Calculate f(S)
2. Initialize the adjacent-color table M.
3. While {stop condition is not met}

3.1 Construct the neighborhood of S, denoted by N(S)

3.2 Calculate the Δ values of all critical one-moves

3.3 Find the best tabu and non-tabu moves with the least Δ value

3.4 If {the aspiration condition is satisfied}

perform the best tabu move,

else

perform the best non-tabu move

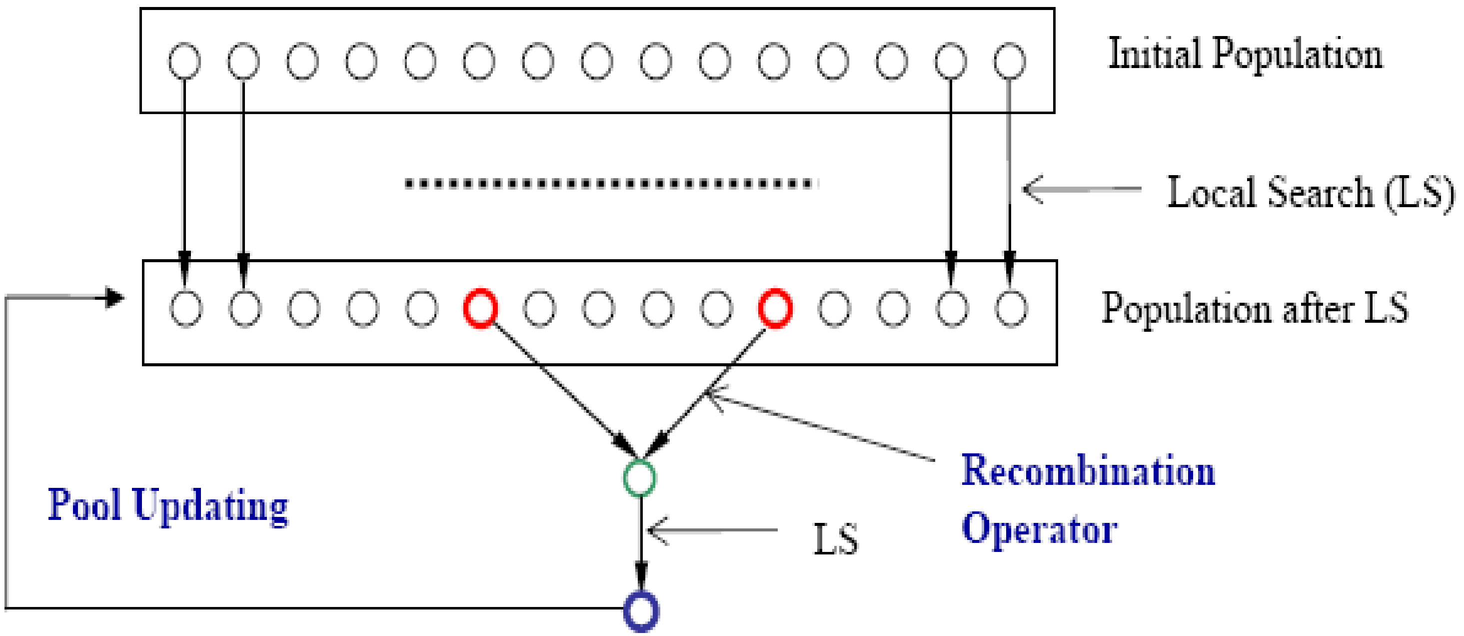
3.5 Update *f* and the adjacent-color table M

End

Hybrid Evolutionary Algorithm

TS with Path Relinking

Main Scheme



Hybrid Evolutionary Algorithm (LS+EA)

Path Rellinking Algorithm

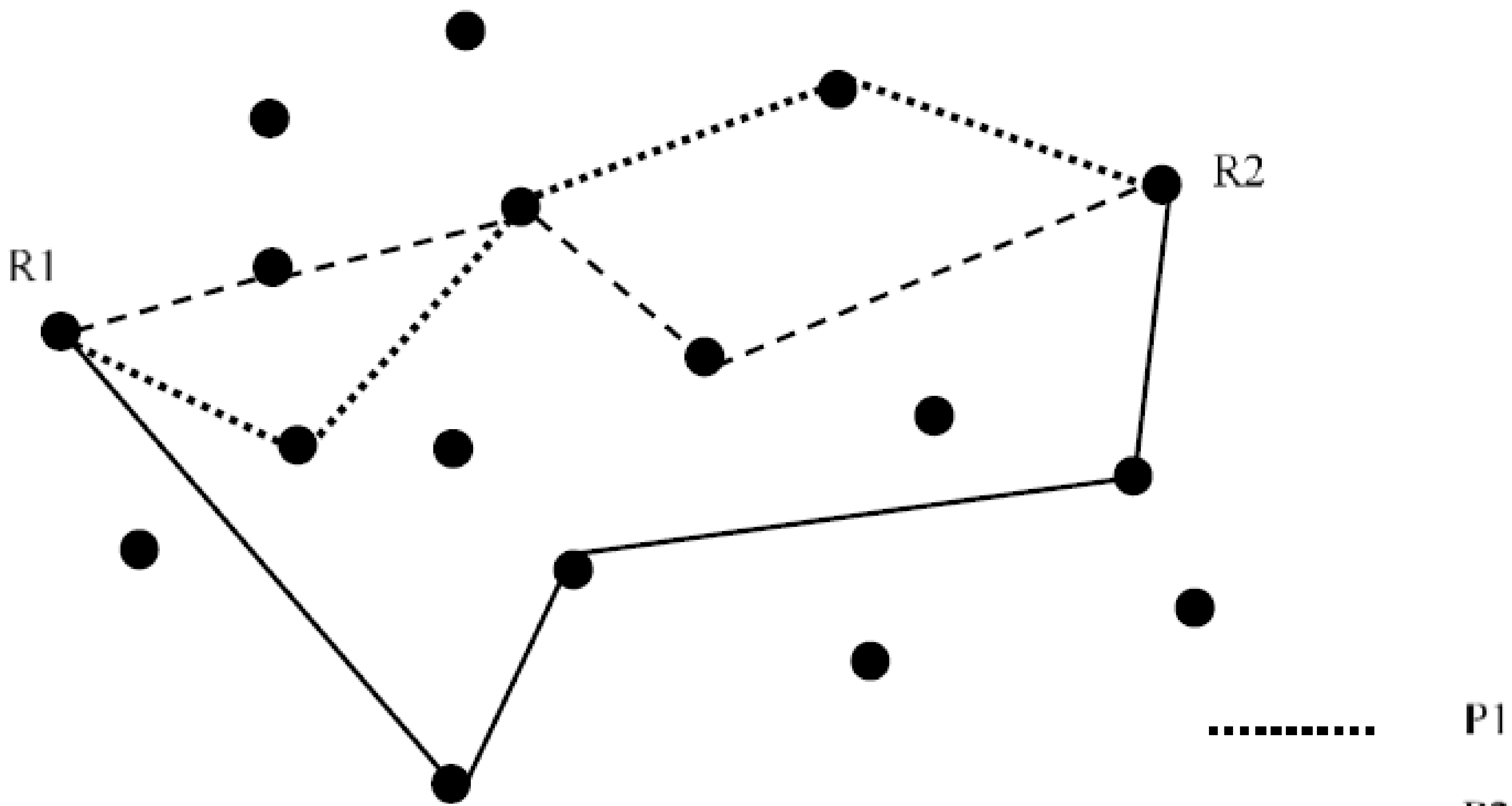
Path-relinking (PR) was originally proposed by Fred Glover as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search.

Starting from one or more elite solutions, paths in the solution space leading towards other elite solutions are generated and explored in the search for better solutions.

To generate paths, moves are selected to introduce attributes that are contained in the elite guiding solution into the current solution.

Path-relinking may be viewed as a strategy that seeks to incorporate attributes of high-quality solutions, by favoring these attributes in the selected moves

Path Rellinking Crossover Operator



Path Rellinking Crossover Operator

Algorithm 3 . Pseudo-Code of Path-Relinking for p-Center

Input: the current GRASP solution S^c, the guiding solution S^e, parameter β
Output: the best solution S^r found by path-relinking
$S^r \leftarrow S^c$, $Q \leftarrow \emptyset$
$S^{c1} \leftarrow S^c \setminus (S^c \cap S^e)$, $S^{e1} \leftarrow S^e \setminus (S^c \cap S^e)$
for i from 1 to $\beta \times d(S^c, S^e)$ **do**
    for each vertex $v_c \in S^{c1}$ **do**
        for each vertex $v_e \in S^{e1}$ **do**
            $S^t \leftarrow S^r \setminus \{v_c\}$, $S^t \leftarrow S^r \cup \{v_e\}$, $Q \leftarrow Q \cup \{(S^t, v_c, v_e)\}$
        end for
    end for
    $(S^r, v_c, v_e) \leftarrow \text{argmin}\ \{f(S) | (S, v_i, v_j) \in Q\}$
    $S^{c1} \leftarrow S^{c1} \setminus \{v_c\}$, $S^{e1} \leftarrow S^{e1} \setminus \{v_e\}$, $Q \leftarrow \emptyset$
end for
$S^r \leftarrow TabuSearch(S^r)$

$$d(S^i, S^j) = p - |S^i \cap S^j|.$$

Path Rellinking Algorithm

Let Sc1 and Se1 be the sets that contain different vertices between Sc and Se, respectively.

Let Q be a set that contains specific trials defined as (S, vi, vj), where S is a solution of the problem, and vi and vj are two different vertices.

For each vertex vc in Sc1 and each vertex ve in Se1, it removes vc out of Sr and adds ve into Sr. The resulting solution St and the corresponding vertices vc and ve are composed as an element (St, vc, ve) which is added to Q (line 8).

The best solution in the element (S, vi, vj) of Q (i.e., the solution with the least increment of the objective function value) is recorded in Sr (line 11).

Then it removes vc and ve out of Sc1 and Se1, respectively. This procedure repeats $\beta \times d(Sc, Se)$ times, where β is a parameter that represents the ratio of the position of Sr in the path from the initial solution to the guiding solution.

Path Rellinking Crossover Operator

β is an important parameter that controls the distance between the intermediate solution and the starting solution. It represents the diversification effect of path-relinking.

It is reasonable to consider that too large or small value of β may not provide a proper dose of diversification because it produces a solution that are too close to the guiding solution or the starting solution.

Path Rellinking Tabu Search Algorithm

Algorithm 1 . Pseudo-Code of GRASP/PR Algorithm for p-Center

Input: problem instance
Output: the best found solution S^*
$P \leftarrow \emptyset$
while stopping condition is not satisfied **do**
  $S \leftarrow GreedyRandomized()$
  $S^t \leftarrow TabuSearch(S)$
  if P is full **then**
    Select an elite solution $S^e \in P$ at random
    $S^r \leftarrow PathRelinking(S^t, S^e)$
    if $S^r \notin P$ and $f(S^r) \leqslant \max\{f(S)|S \in P\}$ **then**
      Let $P' = \{S|S \in P, f(S) \geqslant f(S^r)\}$
      Let $S^w \in P'$ be the most similar solution to S^r, i.e., $S^w = \text{argmin}\{d(S, S^r)|S \in P'\}$
      If there is more than one S^w, randomly select one as S^w
      $P \leftarrow P \cup \{S^r\}$, $P \leftarrow P \setminus \{S^w\}$
    end if
  else
    if $S^t \notin P$ **then**
      $P \leftarrow P \cup \{S^t\}$
    end if
  end if
end while
$S^* \leftarrow \text{argmin}\{f(S)|S \in P\}$

Instances

'http://people.brunel.ac.uk/~mastjjb/jeb/orlib/pmedinfo.html, Jan. 2017.

'http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/, Jan. 2017.

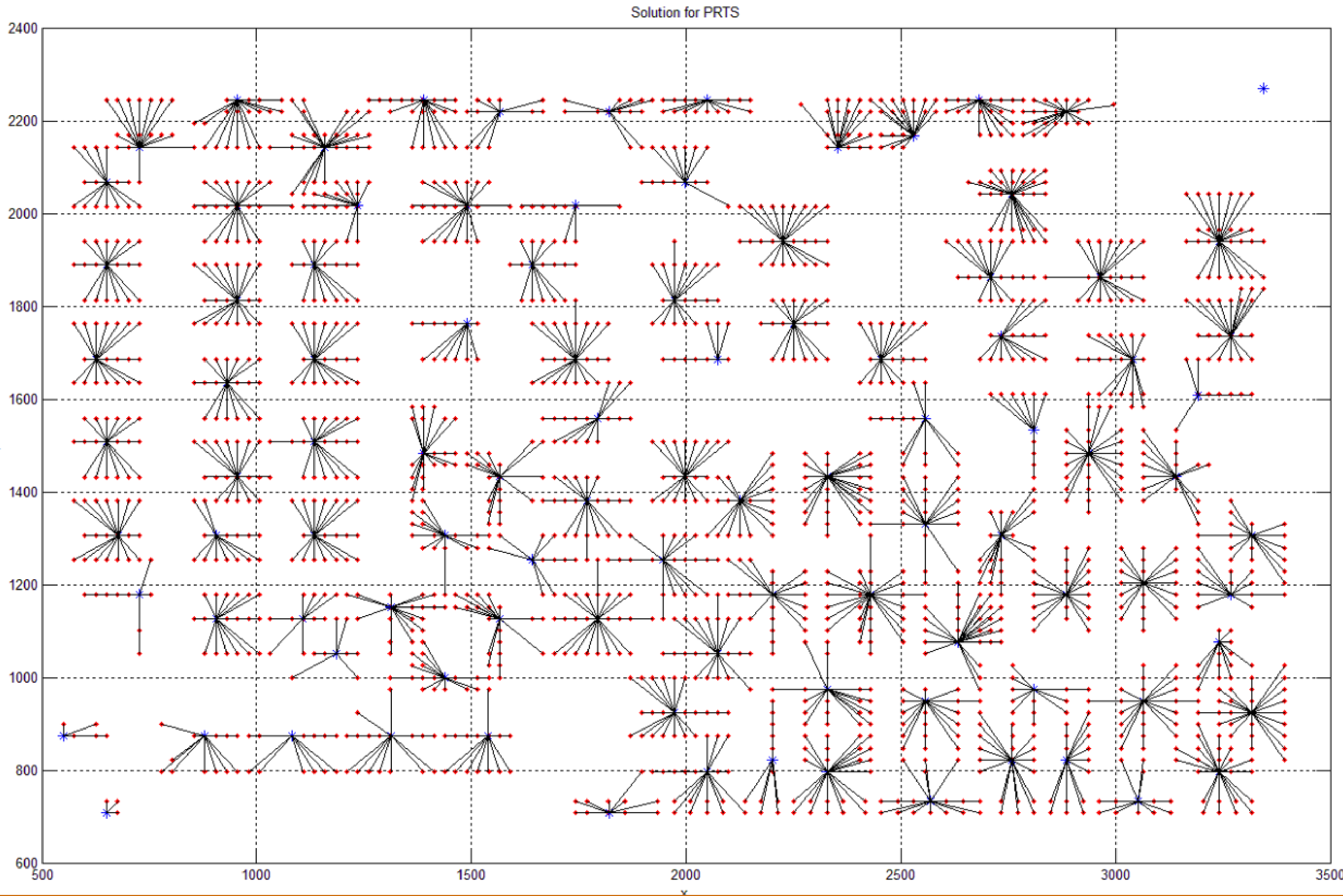
测试结果

➢每个算例都可以算到目前的国际论文中已知的最优解

➢并且可以改进若干算例的最优解

文件名	n	p	PBS-1 Opt	PRTS Best	PBS-1 CPU	PRTS CPU	PRTS HIT
u1817	1817	10	457.91	457.91	5316.6	604.53	100%
u1817	1817	20	309.01	309.01	10243	4068.06	100%
u1817	1817	30	240.99	240.99	1605.5	1239.97	100%
u1817	1817	40	209.46	$\boxed{209.45}$	193.7	308.29	100%
u1817	1817	50	184.91	184.91	1128.9	471.94	100%
u1817	1817	60	162.65	$\boxed{1062.64}$	837.3	469.43	100%
u1817	1817	70	148.11	148.11	191.8	19.66	100%
u1817	1817	80	136.8	136.79	127.5	12.42	100%
u1817	1817	90	129.54	$\boxed{129.51}$	2963.5	3859.05	100%
u1817	1817	100	127.01	$\boxed{126.99}$	146.4	2.35	100%
u1817	1817	110	109.25	109.25	13772.4	6954.89	80%
u1817	1817	120	107.78	$\boxed{107.76}$	80.1	5.25	100%
u1817	1817	130	107.75	107.75	11.2	7.04	100%
u1817	1817	140	101.61	101.61	4949.3	30.95	100%
u1817	1817	150	101.6	$\boxed{92.44}$	314	1236.55	20%

算例U1817.tsp的运行结果



扩展

- P-center问题再增加/修改一些约束条件或者改变目标函数可以得到很多变形的问题，比如：

1. 每个服务结点最多只能服务一定数量的用户
2. 建立服务设施需要一定的费用
3. 所有服务边距离的总和最小
4. 在特定距离内满足所有用户的需求，求最小服务设施数（覆盖问题)

Thanks!