VRPTW

--Vehicle Routing Problem with Time Windows

SmartLab-李云皓

目录
CONTENTS

Part 01

问题介绍

1.1

VRP

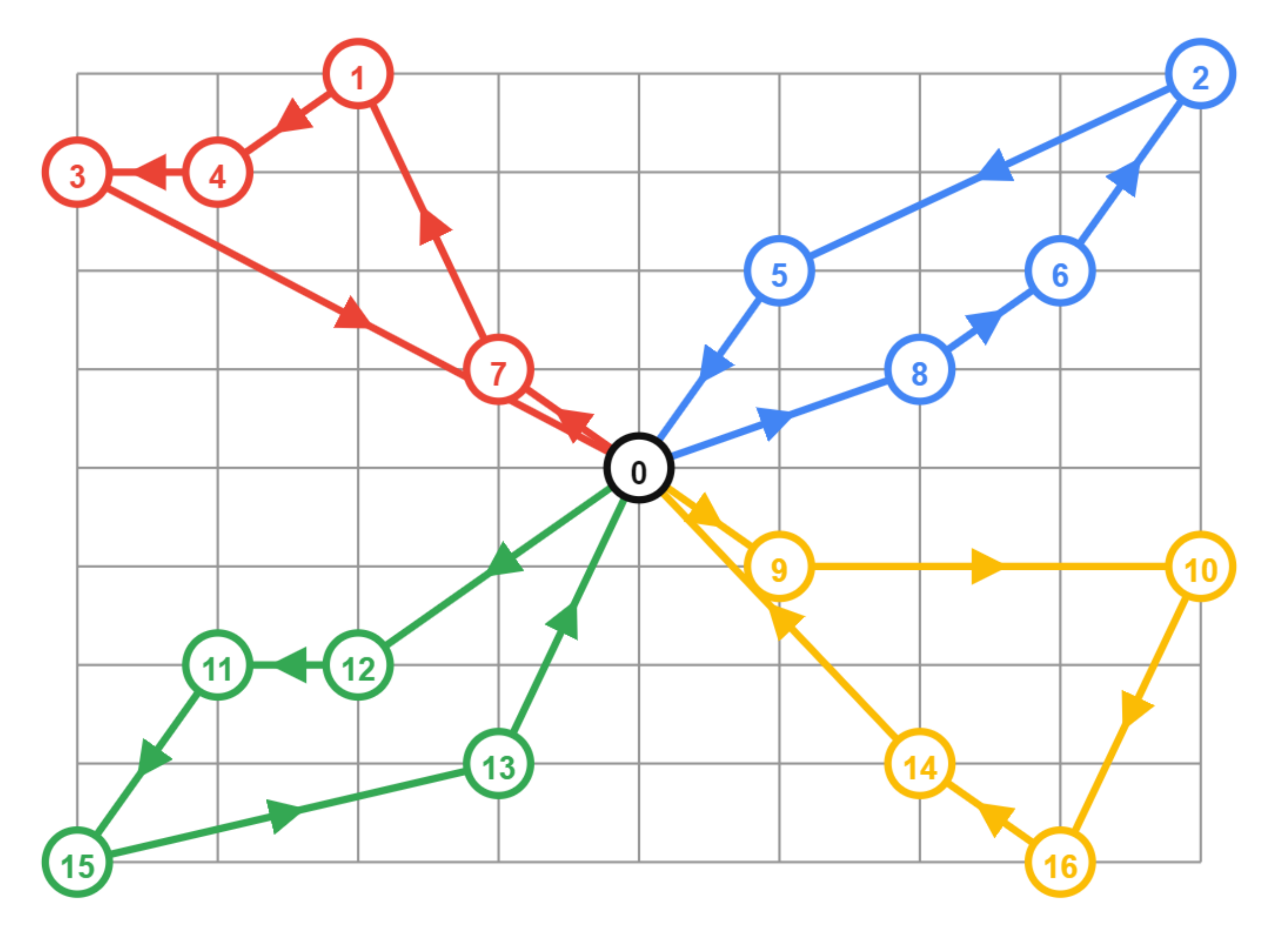
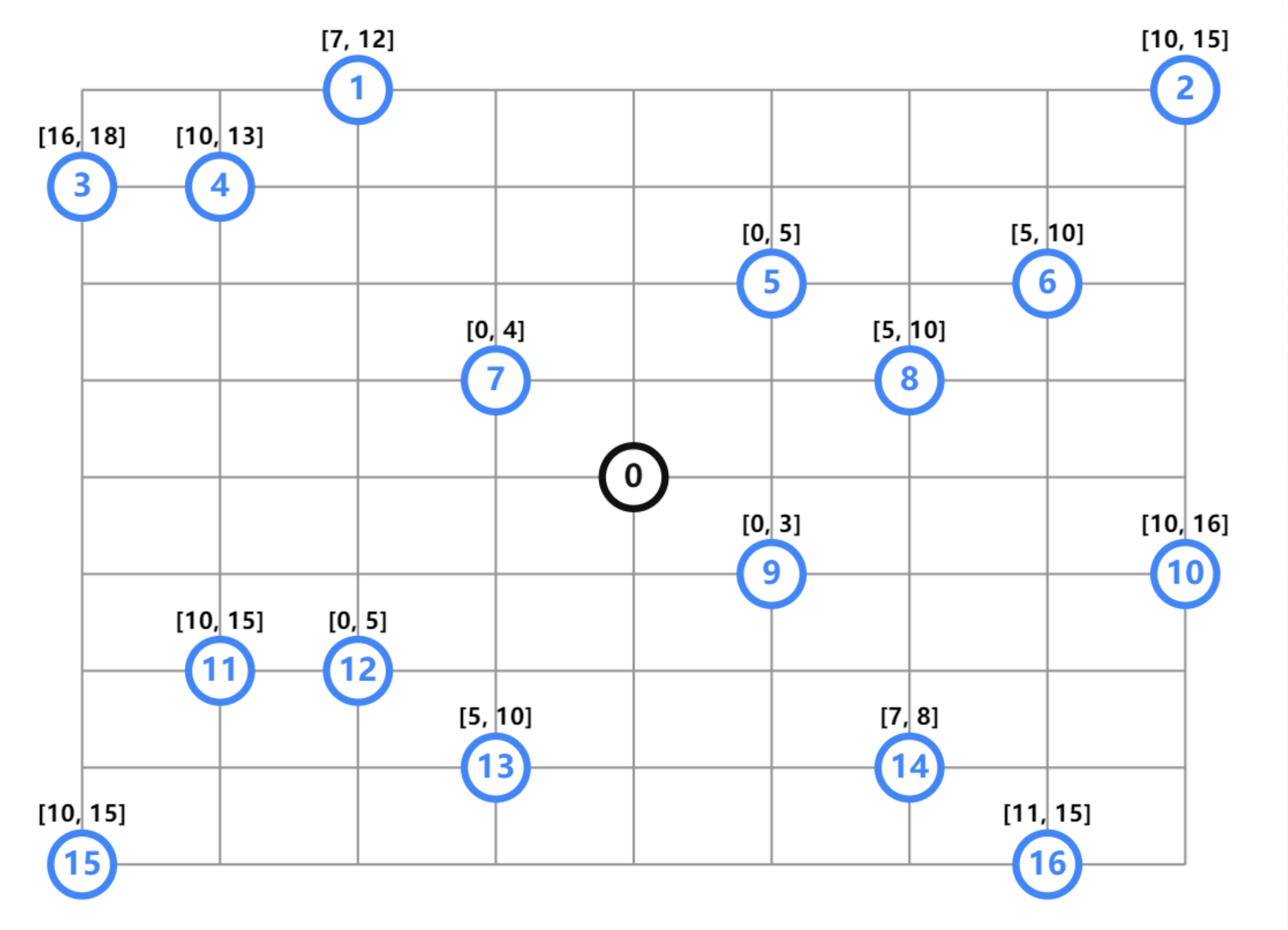


Fig.1 [1]

1.1 VRP

一个起点
每个点只能被访问一次
一辆车负责一条路线
每一个都看做一个TSP
车辆的容量约束
路径数量最少的方案
路径长度最小的方案
车辆最少的方案

1.2 TW



[7, 12]
[10, 15]
[16, 18]
[10, 13]
[0, 5]
[5, 10]
[0, 4]
[5, 10]
[0, 3]
[10, 16]
[10, 15]
[0, 5]
[5, 10]
[7, 8]
[10, 15]
[11, 15]
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Fig.2 [2]

1.3

应用

配送中心配送、
公共汽车路线制定、
信件和报纸投递、
航空和铁路时间表安排、
工业废品收集等[3]
上学和放学？

1.3 VRP分支[4]

CVRP：Capacitated VRP,
限制配送车辆的承载体积、重量等。

VRPTW：VRP with Time Windows,
客户对货物的送达时间有时间窗要求。

VRPPDP：VRP with Pickup and Delivery,
车辆在配送过程中可以一边揽收一边配送，在外卖O2O场景中比较常见(PDPTW)。

1.3 VRP分支[4]

MDVRP： Multi-Depot VRP,
配送网络中有多个仓库，同样的货物可以在多个仓库取货。

OVRP：Open VRP,
车辆完成配送任务之后不需要返回仓库。

VRPB： VRP with backhauls,
车辆完成配送任务之后回程取货。

Part 02

数学模型

2.1

四要素

目标：最小化路径数量
最小化总时间

决策：每辆车的路径

已知：
任意两点间的行驶时间Cij
每一个点的服务时间s_v
每一个点的配送容量q_v
每辆车的容量Q
节点的时间窗$[e_v,l_v]$

约束：

2.2

约束

每个客户只能被一辆车服务一次，
车辆必须从配送中心0点出发
每辆车都必须停留在配送中心n+1
车辆的容量约束
顾客以及仓库的时间窗约束

2.3 目标

最小化路径数量

min	$|K|$

2.4

约束

每个客户只能被一辆车服务一次

$\sum_{k\in K}\sum_{j\in N-\{i\}} x_{ijk} = 1$	$\forall i \in N$
$\sum_{i\in N-\{j\}} x_{ijk} - \sum_{i\in N-\{j\}} x_{jik} = 0$	$\forall k \in K, \forall j \in N$

2.4

约束

车辆必须从配送中心0点出发
每辆车都必须回到0点
时间约束
消除子环

$\sum_{j \in N} x_{0jk} = 1$	$\forall k \in K$
$\sum_{i \in N} x_{i,n+1,k} = 1$	$\forall k \in K$
$T_{ik} + s_i + C_{ij} - T_{jk} \leq (1 - x_{ijk})M_{ij}$	$\forall k \in K, (i,j) \in A$
$e_i \leq T_{ik} \leq l_i$	$\forall k \in K, i \in V$

2.4 约束

车辆的容量约束

$\sum_{i \in N} q_i \sum_{j \in N-\{i\}} x_{ijk} \leq Q$	$\forall k \in K$

2.5 模型

min	$\lvert K \rvert$	
s.t.	$\sum_{k\in K}\sum_{j\in N-\{i\}} x_{ijk} = 1$	$\forall i \in N$
	$\sum_{i\in N-\{j\}} x_{ijk} - \sum_{i\in N-\{j\}} x_{jik} = 0$	$\forall k \in K, \forall j \in N$
	$\sum_{j\in N} x_{0jk} = 1$	$\forall k \in K$
	$\sum_{i\in N} x_{i,n+1,k} = 1$	$\forall k \in K$
	$T_{ik} + s_i + C_{ij} - T_{jk} \le (1 - x_{ijk})M_{ij}$	$\forall k \in K, (i,j) \in A$
	$e_i \le T_{ik} \le l_i$	$\forall k \in K, i \in V$
	$\sum_{i\in N} q_i \sum_{j\in N-\{i\}} x_{ijk} \le Q$	$\forall k \in K$
	$x_{ijk} \in \{0,1\}$	$\forall k \in K, (i,j) \in A$

Part 03

优化路径数

3.1主程序
deleteRoute

3.1

初始化

初始化一个可行解a
每一个顾客由一个车进行服务
路线之间彼此没有交集

3.1

移除环

从已经的几条路径里面任意移除一个环
这时a变成了一个局部解(partial solution)
被移出去的节点构成集合EP（ejection pool)
ejection pool用来保存没有被服务的点

3.1

插入一个点

从EP中挑一个点V尝试插入a中
不违反时间窗约束、容量约束
如果有合法的位置
每一个合适的插入位置都可以形成一个可行解a’
a’ 可以构成一个集合Ninset(a,v)
从Ninset(a,v)中任意找一个a’ 来更新a
进行下一轮迭代

3.1

无合法的位置

使用Squeeze（v,a）再次进行插入v
暂时接受一个非法的局部解
然后尝试通过调整使局部解合法
调整成功进行下一次迭代
调整失败只能丢弃一些点

3.1

调整失败

如果调整失败，就必须扔掉一些点
这时候可能有多种丢弃方案
V也可能被丢弃出去
丢弃后每个方案对应合法一个局部解a′
a′构成的集合$N_{ej}(v,a)$
需要评估每一个丢弃方案的优劣
之后进行下一轮迭代

3.1

DeleteRoute-1

Procedure DELETEROUTE(σ)
begin
Select and remove a route randomly from σ;
Initialize *EP* with the customers in the removed route;
Initialize all penalty counters $p[v] := 1$ $(v = 1, \ldots, N)$;
while $EP \neq \emptyset$ and $time < maxTime$ **do**
  Select and remove customer v_{in} from *EP* with the LIFO strategy;
  if $\mathcal{N}^{fe}_{insert}(v_{in}, \sigma) \neq \emptyset$ **then**
    Select $\sigma' \in \mathcal{N}^{fe}_{insert}(v_{in}, \sigma)$ randomly; Update $\sigma := \sigma'$;
  else
    $\sigma := \text{SQUEEZE}(v_{in}, \sigma)$;
  endif

3.1 DeleteRoute-2

if v_{in} is not included in σ **then**

Set $p[v_{in}] := p[v_{in}] + 1$;

Select $\sigma' \in \mathcal{N}_{EJ}^{fe}(v_{in}, \sigma)$ such that $P_{sum} = p[v_{out}^{(1)}] + \ldots + p[v_{out}^{(k)}]$ is minimized; Update $\sigma := \sigma'$;

Add ejected customers $\{v_{out}^{(1)}, \ldots, v_{out}^{(k)}\}$ to EP;

$\sigma :=$ PERTURB(σ);

endif

endwhile

if $EP \neq \emptyset$ **then** Restore σ to the input state;

return σ;

end

3.2 邻域

3.2

邻域动作2-opt

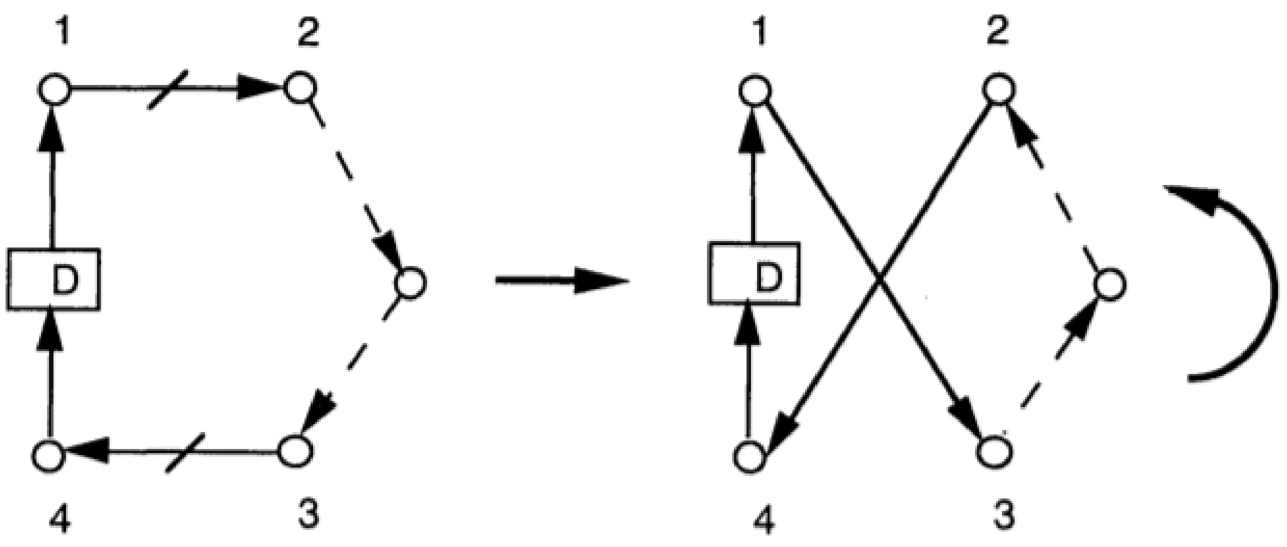
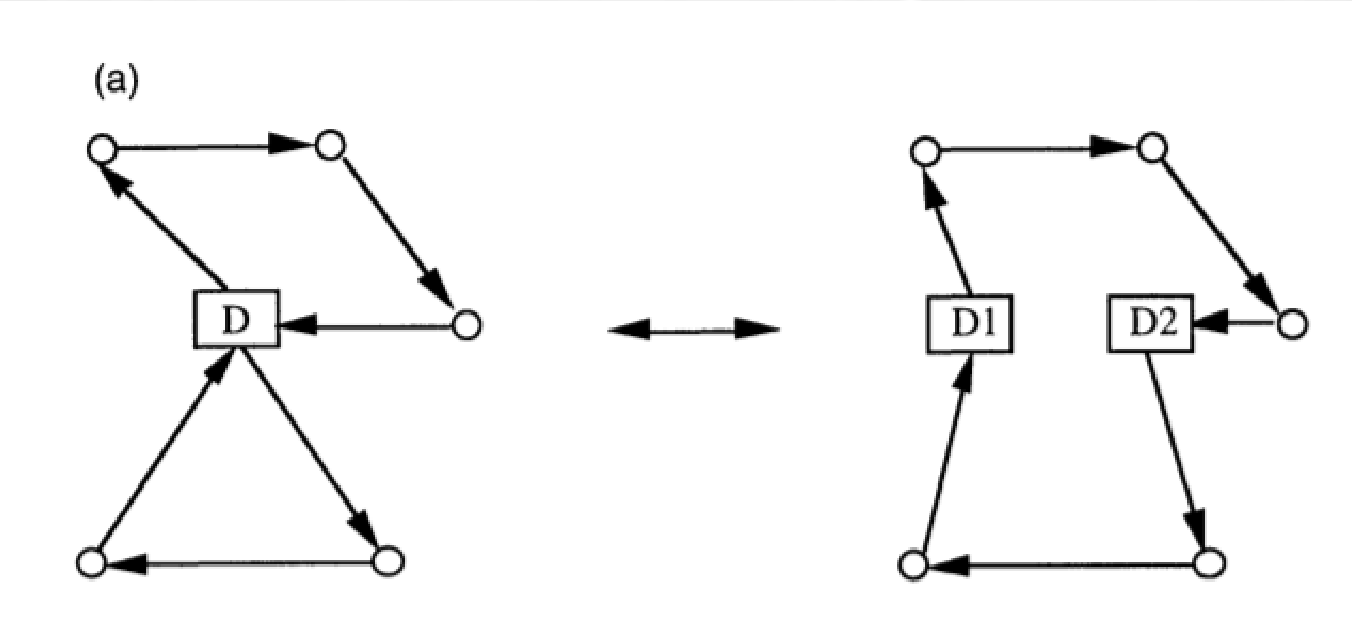


FIG. 1. *Exchange of links* (1, 2), (3, 4) *for links* (1, 3), (2, 4).

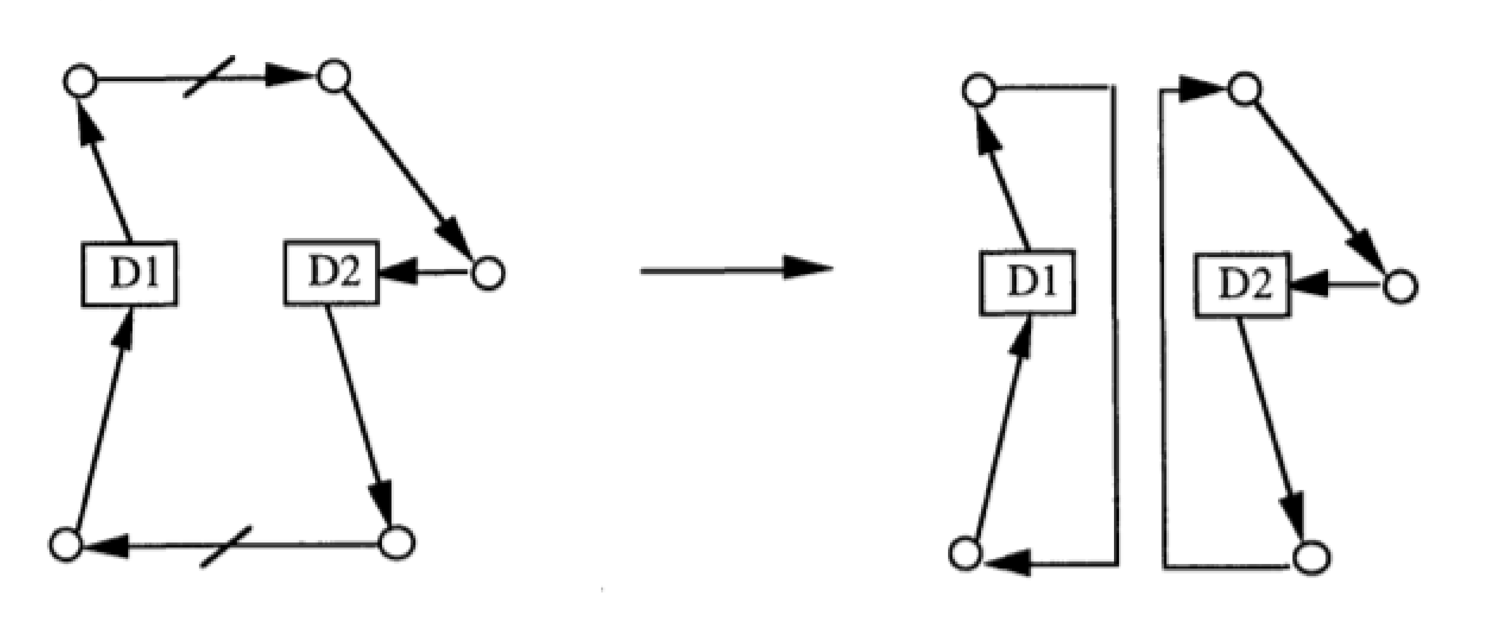
选择两个点，将两个点之间的节点顺序颠倒
适用于没有时间窗的问题
对于有时间窗的问题，此过程产生了不合法解

3.2 邻域动作2-opt*



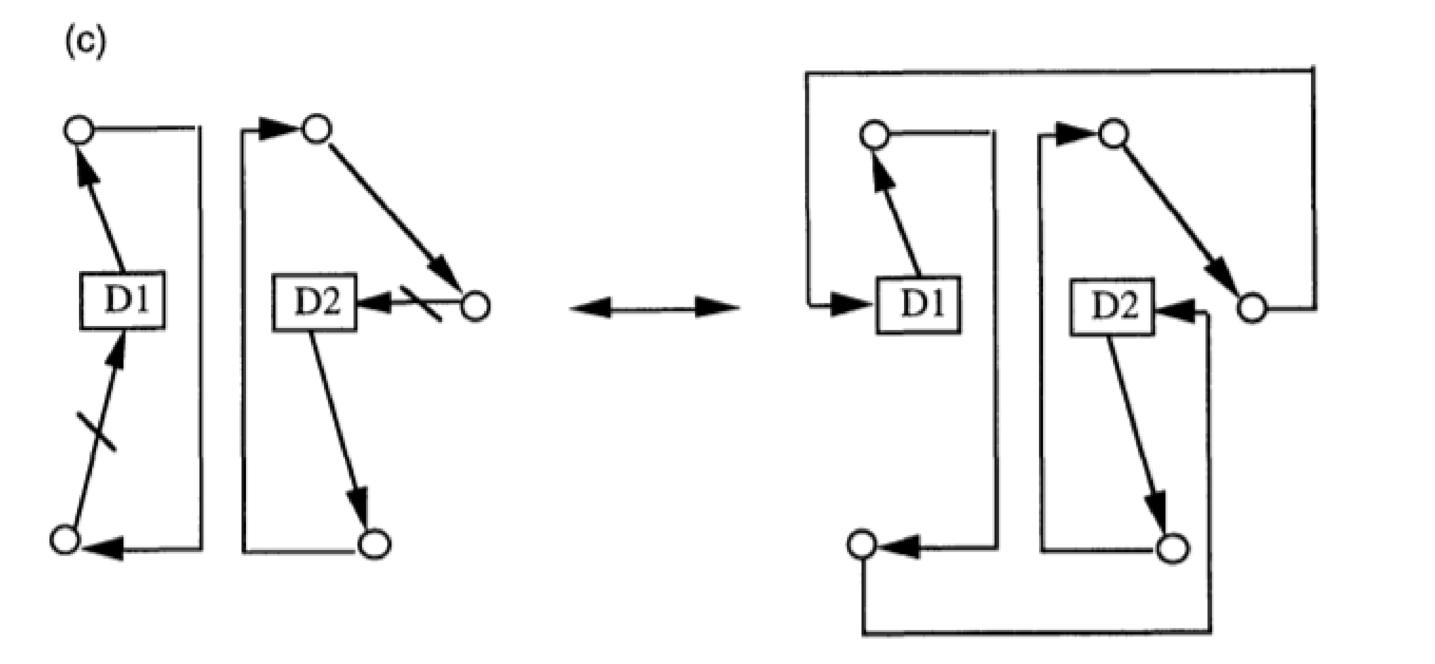
等价转换，添加depot

3.2 邻域动作2-opt*



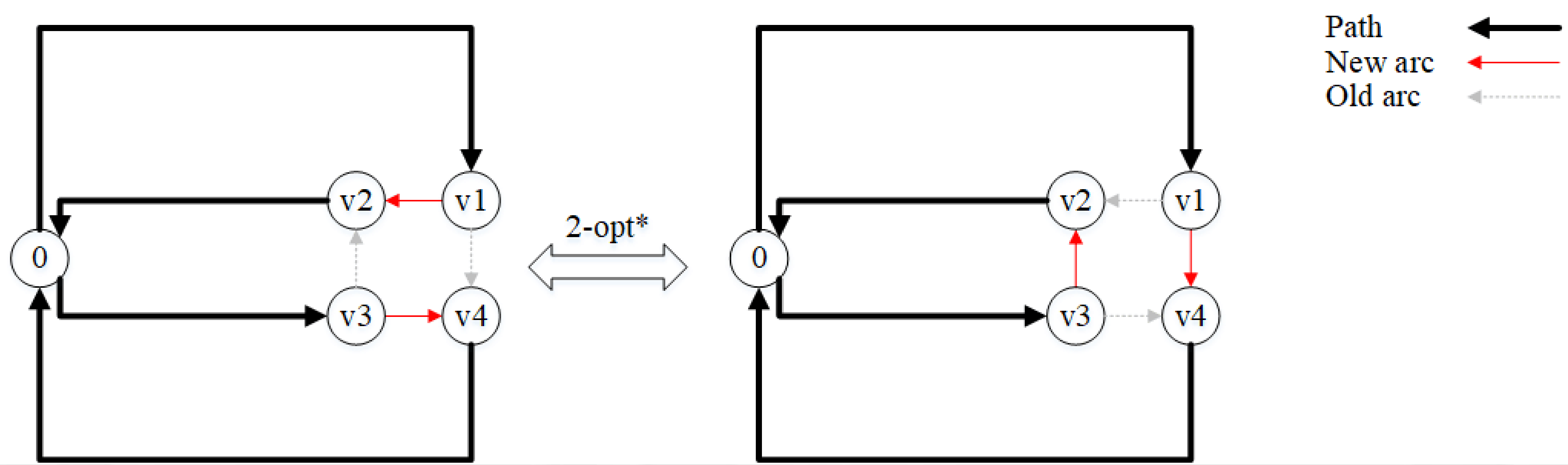
去掉两条路径，将回路一分为二
如图各自形成一个环路

3.2 邻域动作2-opt*



去掉最后回到depot的边
交换每条路径回去的仓库
将两个环路合并

3.2 邻域动作2-opt*



3.2

邻域动作 relocate

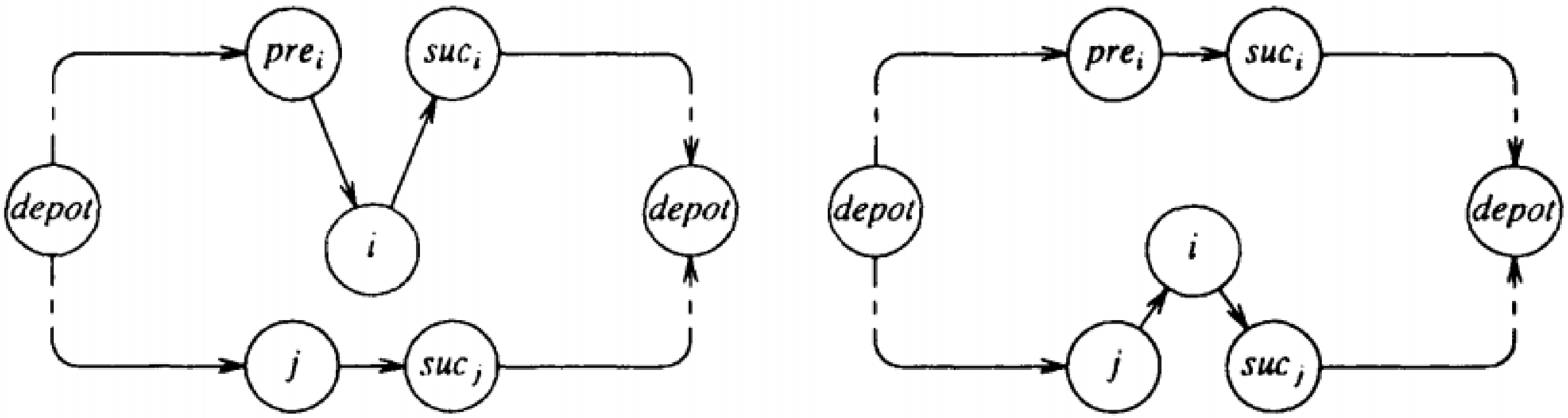


Figure 10.4 A relocation

relocate

3.2 邻域动作 exchange

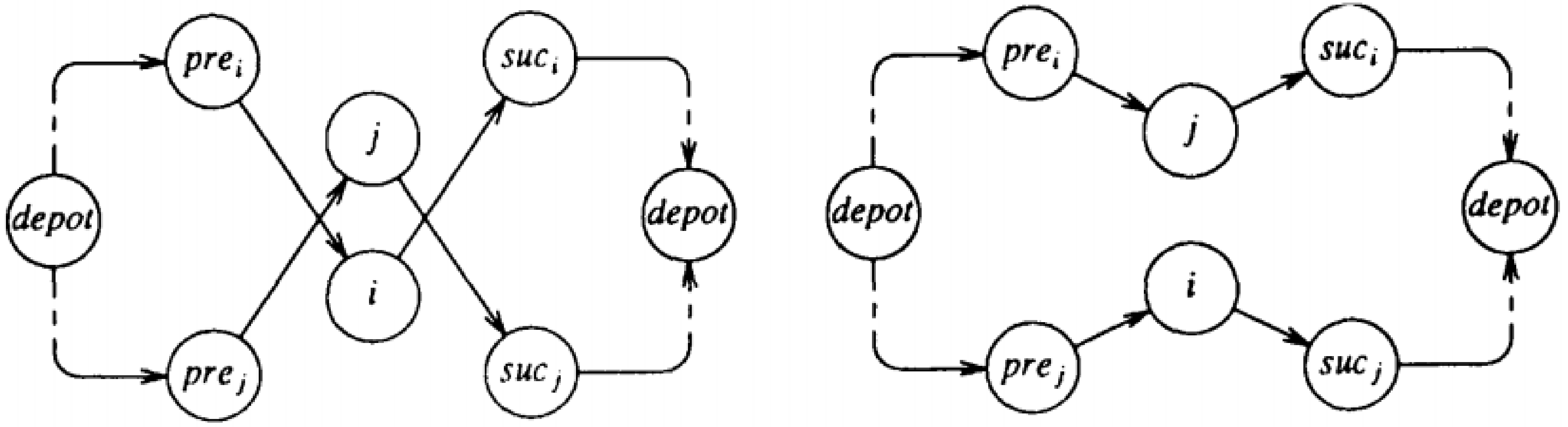


Figure 10.6 An exchange

exchange

3.2 限制邻域

为了缩小邻域
我们只考虑选定节点的一定范围之内的节点
论文中选取了距离节点前100近的节点

3.3 邻域评估

3.3

Penalty Function

$$F_p(\sigma) = P_c(\sigma) + \alpha \cdot P_{tw}(\sigma).$$

Fp(a): 评估局部解优度的函数（打分）
Pc(a): 定义为每条路超过容量的值求和
Ptw(a):每一条路超过的时间约束
α采用0.99倍调整

3.3

Ptw(a)

$$\tilde{a}_{v_0} = e_0, (\tilde{a}'_{v_0} = e_0),$$

$$\tilde{a}'_{v_i} = \tilde{a}_{v_{i-1}} + s_{v_{i-1}} + c_{v_{i-1}v_i} \quad (i = 1, \dots, n+1),$$

$$\begin{cases} \tilde{a}_{v_i} = \max\{\tilde{a}'_{v_i}, e_{v_i}\} & \text{if } \tilde{a}'_{v_i} \le l_{v_i} \\ \tilde{a}_{v_i} = l_{v_i} & \text{if } \tilde{a}'_{v_i} > l_{v_i} \end{cases} \quad (i = 1, \dots, n+1). \tag{4}$$

$$TW(r) = \sum_{i=0}^{n+1} \max\{\tilde{a}'_{v_i} - l_{v_i}, 0\}. \tag{5}$$

$$P_{tw}(\sigma) = \sum_{r=1}^{m} TW(r).$$

3.4 ejection评估

3.4 ejection

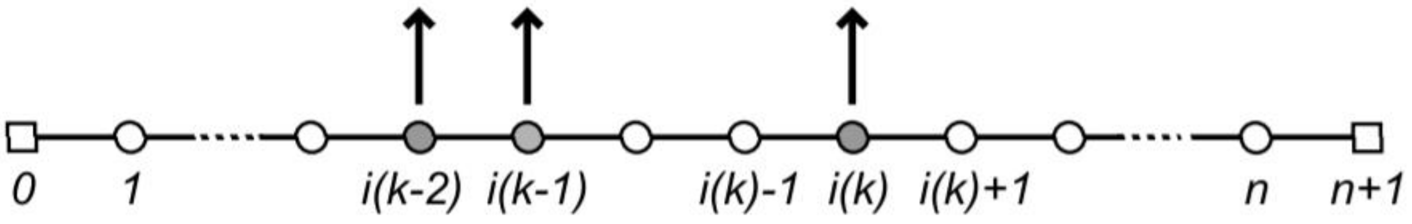


Fig. 3. An example of the lexicographic ejections. Customer nodes indicated by gray circles are ejected. If $k_{max} = 3$, customer nodes are ejected in the following order: $\{1\}, \{1, 2\}, \{1, 2, 3\}, \{1, 2, 4\}, \ldots, \{1, 2, n\}, \{1, 3\}, \{1, 3, 4\}, \ldots, \{1, n-1, n\}, \{2\}, \ldots$. For example, when customers $\{5, 6, 9\}$ are ejected, we gain $k = 3$, $i(k) = 9$, $i(k-1) = 6$ and $i(k-2) = 5$.

3.4

P数组

记录每一个插入路径的次数

反映了这个点插入集合的难度

$P_{sum} = P[V_{1out}] + P[V_{2out}] + \ldots + P[V_{kout}]$

EP中的P值之和可以衡量丢弃方案的好坏

3.4

剪枝

- After $i(k)$ is ejected,
  (a) $\sum_{s=1}^{k} p[i(s)] \geq P_{\text{best}}$ (the minimum value of P_{sum} found so far)
- After $i(k)$ $(= j)$ is incremented,
  (b) $l_j < a_j^{\text{new}}$
  (c) $a_j^{\text{new}} = a_j$ and $Q' \leq Q$

丢弃i(k)的时候，超过Psum
i(k)增加的时候，i(k)之前违反约束
i(k)增加的时候，i(k)没有改善结果
进行剪枝

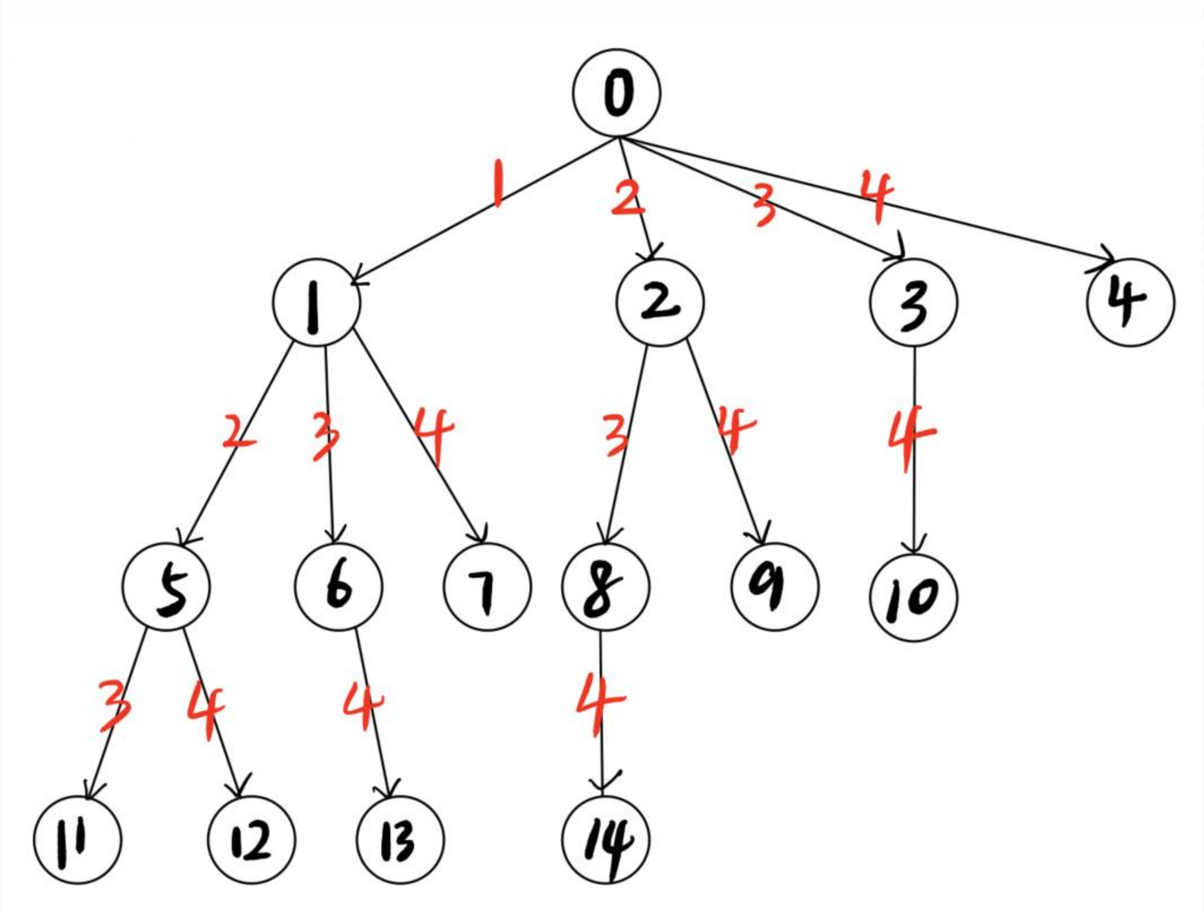
3.4 剪枝

In the case (a), the lower-level lexicographic ejections ($\{i(1), \ldots, i(k), *, *, \ldots\}$) can be pruned because these ejections never improve P_{best} where '$*$' refers to any subsequent customer after $i(k)$. In both cases (b) and (c), the lower-level lexicographic ejections ($\{i(1), \ldots, i(k-1), i(k), *, *, \ldots\}$ ($i(k) > j$) can be pruned because (b) the time window constraint is violated already at customer j, or (c) the ejection of customers $\{i(1), \ldots, i(k-1)\}$ does not accelerate the earliest starting time of customer j (and the capacity constraint need not to be considered).

丢弃i(k)的时候，超过Psum
i(k)增加的时候，i(k)之前违反约束
i(k)增加的时候，i(k)没有改善结果
进行剪枝

3.4

剪枝



E.G. Kmax = 3，N = 4

3.5 Squeeze

3.5 Squeeze

Procedure SQUEEZE(v_{in}, σ)
begin
: Search $\sigma' \in \mathcal{N}_{insert}(v_{in}, \sigma)$ such that $F_p(\sigma')$ is minimum; Update $\sigma := \sigma'$;
: **while** $F_p(\sigma) \neq 0$ **do**
: Randomly select an infeasible route r;
: Search $\sigma' \in \mathcal{N}_r(\sigma)$ such that $F_p(\sigma')$ is minimum;
: **if** $F_p(\sigma') < F_p(\sigma)$ **then** Update $\sigma := \sigma'$;
: **else break**;
: **endwhile**
: **if** $F_p(\sigma) \neq 0$ **then** Restore σ to the input state;
: **return** σ;
end

找一个插入位置最好的，使得容量冲突和时间冲突最小
局部搜索达到最优解
如果可以使得F(a)为0，返回a
否则返回最初的状态

3.6 扰动

3.6 perturb

扰动
在给定的的随机时间内
在邻域的合法解之间进行切换

Part 04

优化时间

4.1 EAMA

4.1 初始化

目标：最小化路径的长度

Npop：种群大小

利用单个解的算法，生成m条路径的合法解

可以看做是基于单个解的第二阶段的目标

4.1

进化

从种群中任取两个不同的解Pa，Pb

将记录当前最好记录为Pa

用交叉算子生成Nch次子代

每次的子代修复以及局部搜索

用子代去更新当前最好

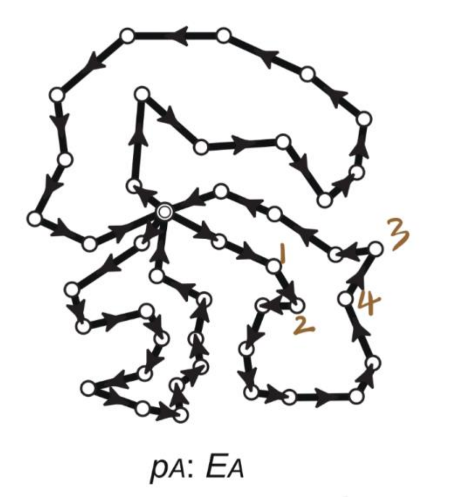
一直到最小化目标实现

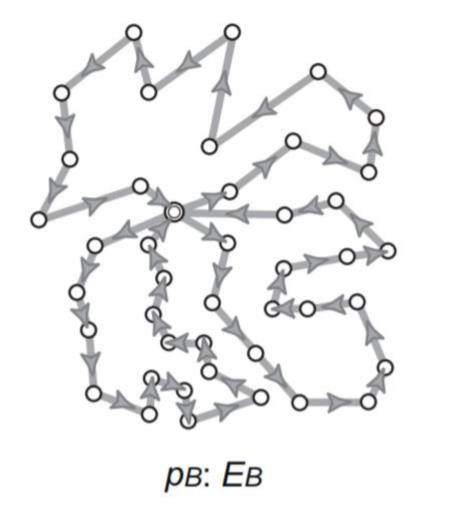
4.1 procedure

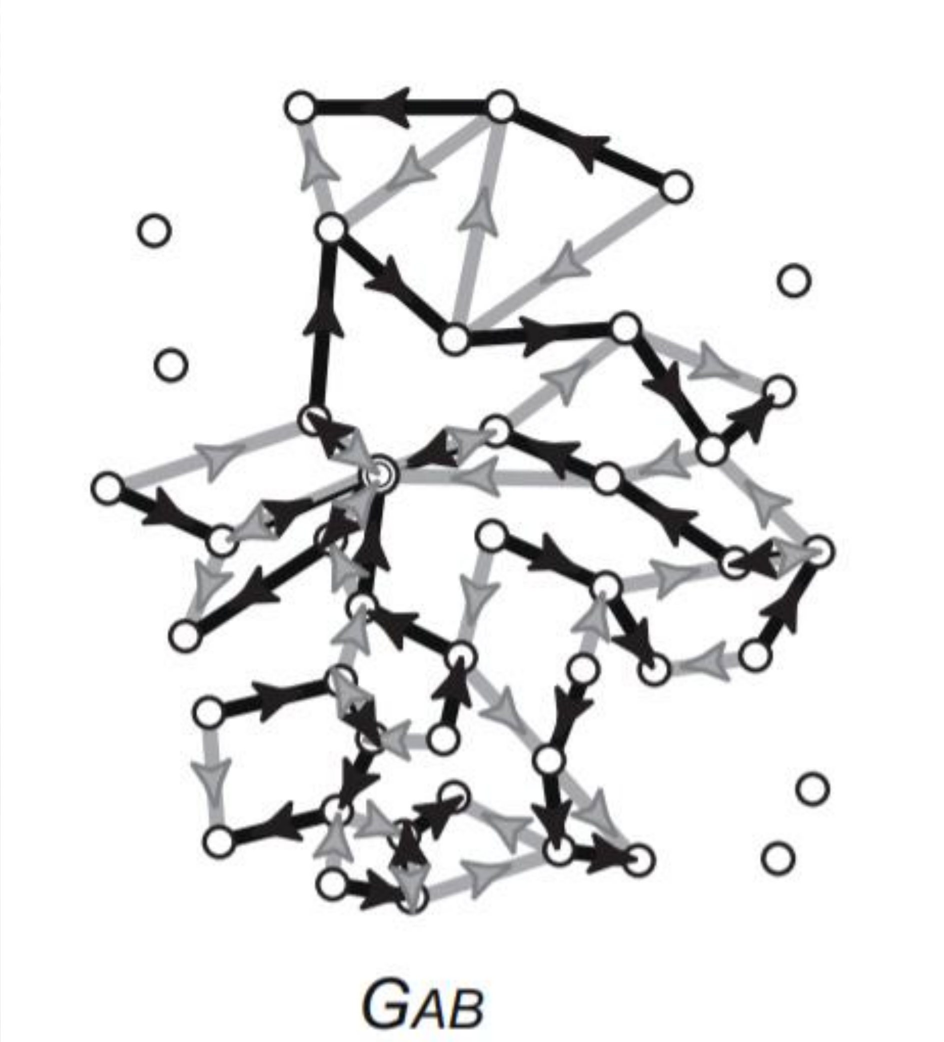
```
Procedure EAMA (N_pop, N_ch)
begin
// Route minimization phase
m := DETERMINE_M();                                   // RMheuristic
for i :=1to N_pop do
    σ_i := GENERATE_INITIAL_SOLUTIONS (m);            // RMheuristic
end for
// Distance minimization phase
repeat
    Let r(i)∈{1,...,N_pop} be a random permutation;
    for i:= 1 to N_pop do
        p_A :=σ_r(i); p_B := σ_r(i+1); (Note: r(N_pop+1) = r(1))
        σ_best :=p_A;
        for j: = 1 to N_ch  do
            σ := EAX (p_A,p_B);                       // crossover
            σ := Repair (σ);                          // repairprocedure
            σ :=  Local_Search(σ);                    // localsearch
            if F(σ) < F(σ_best ) then σ_best :=σ;
        end for
        σ_r(i) := σ_best ;
    end for
until termination condition is satisfied
return the best individual in the population;
end
```

4.2 交叉算子EAX

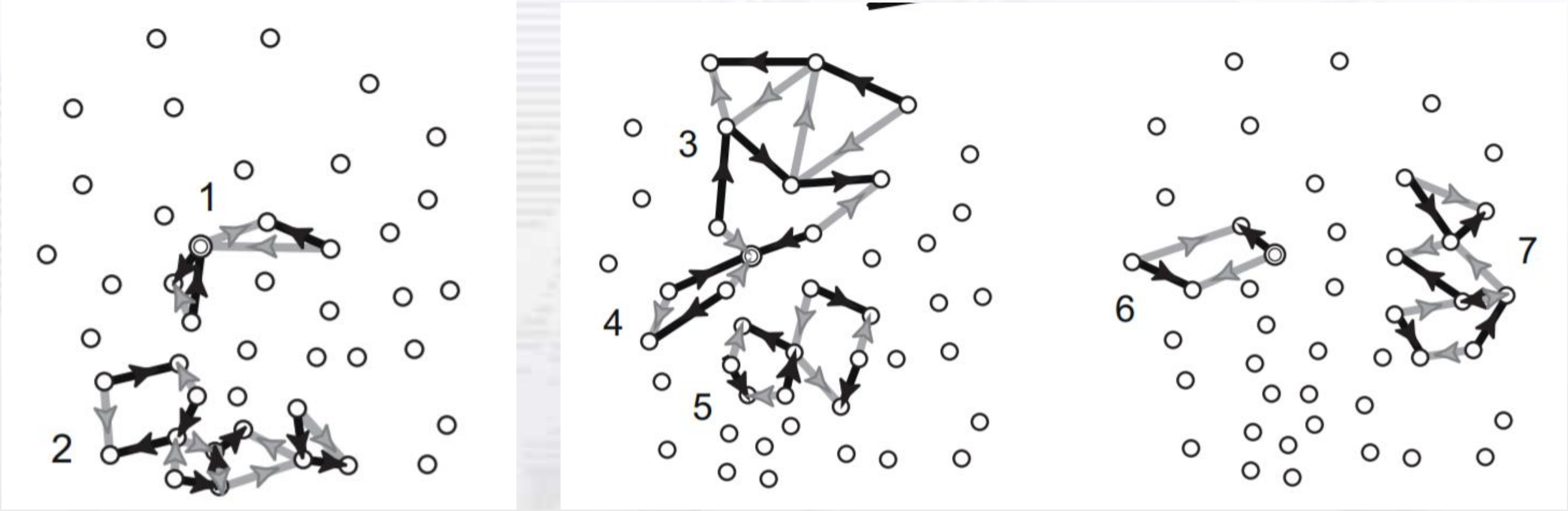
4.2 GAB





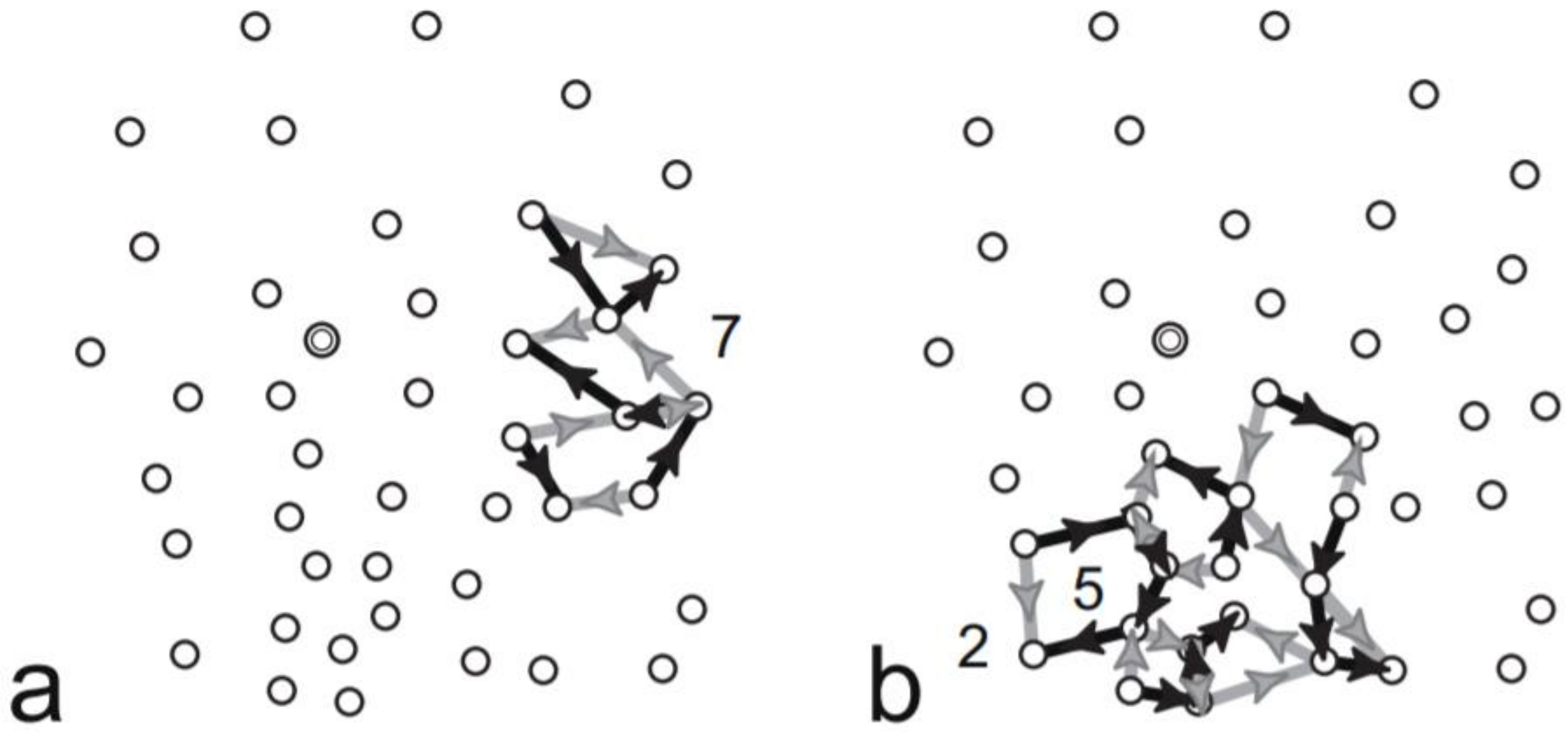


4.2 AB-cycles



4.2

E-sets



E-sets

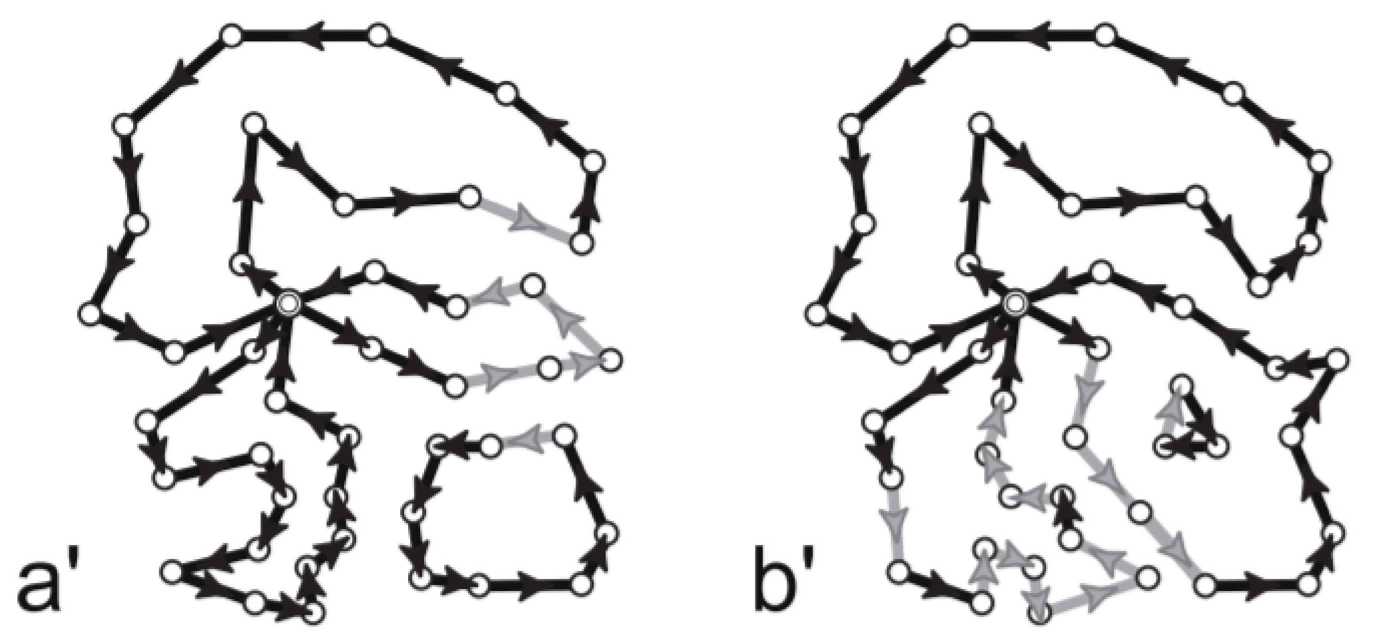
4.2 strategy

Single strategy： 随机选择一个环
Block strategy： 随机选取两个连着的换，必须至少共享一个节点

前者适合 local improvement
后者适合 global improvement

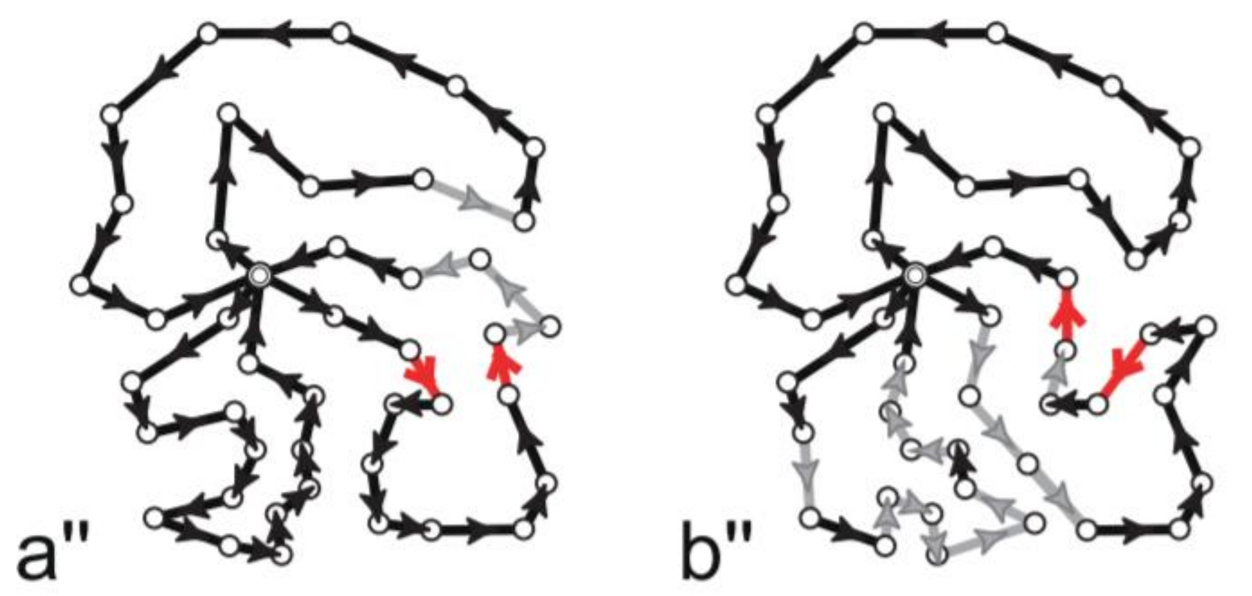
4.2

Inter-solutions



Intermediate solutions

4.2 重组



Offspring solutions

利用2-opt*进行破环重组

4.3 EAX 合理性

4.3

Q&A

1.如何保证环路的闭合

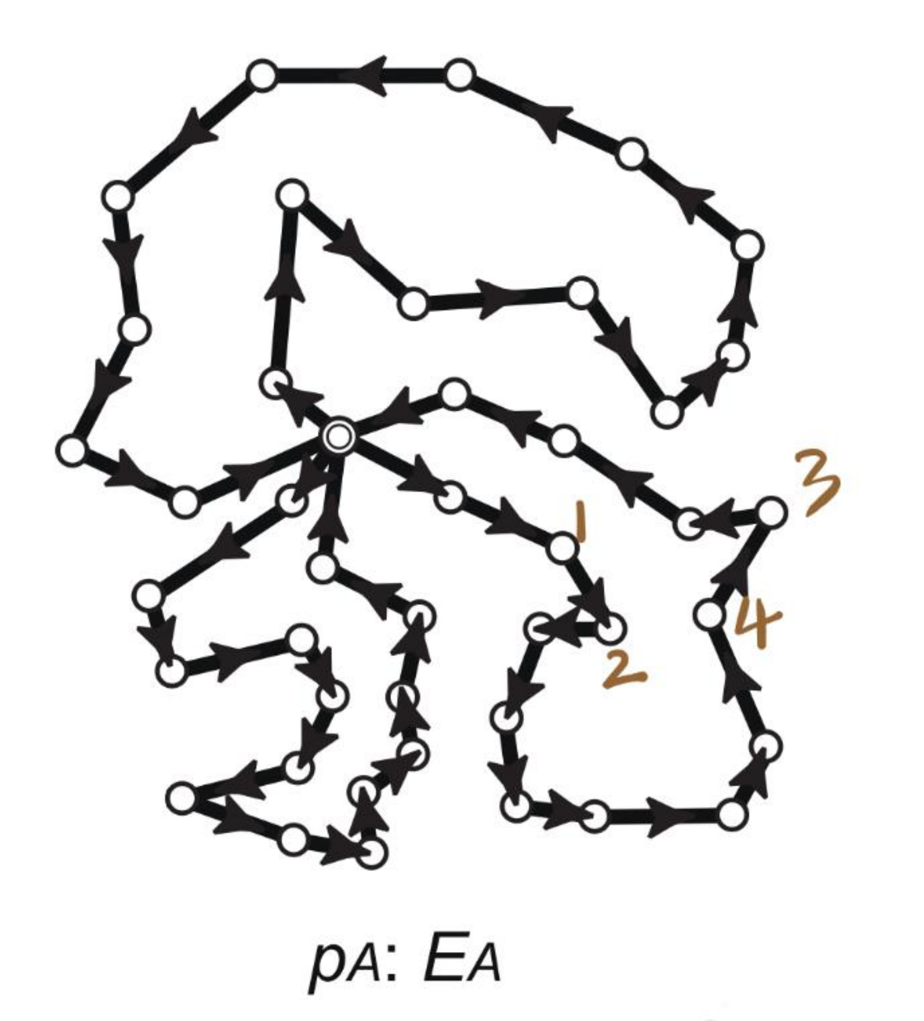
2.$\mathrm{Pa} \cup Pb - (Pa \cap Pb)$

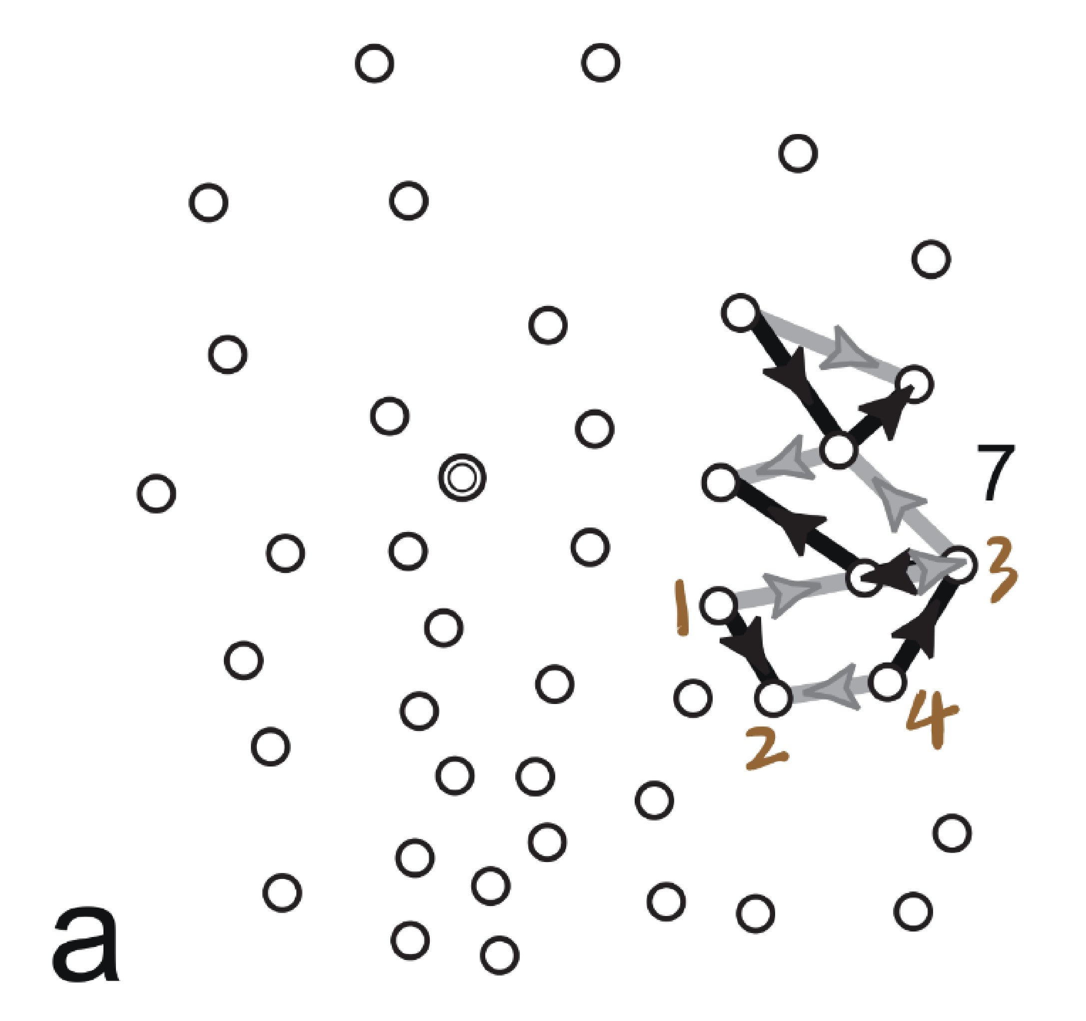
3.$\mathrm{Pa} - (\mathrm{Pa} \cap Eset) + (Pb \cap Eset)$

4.重组的代价

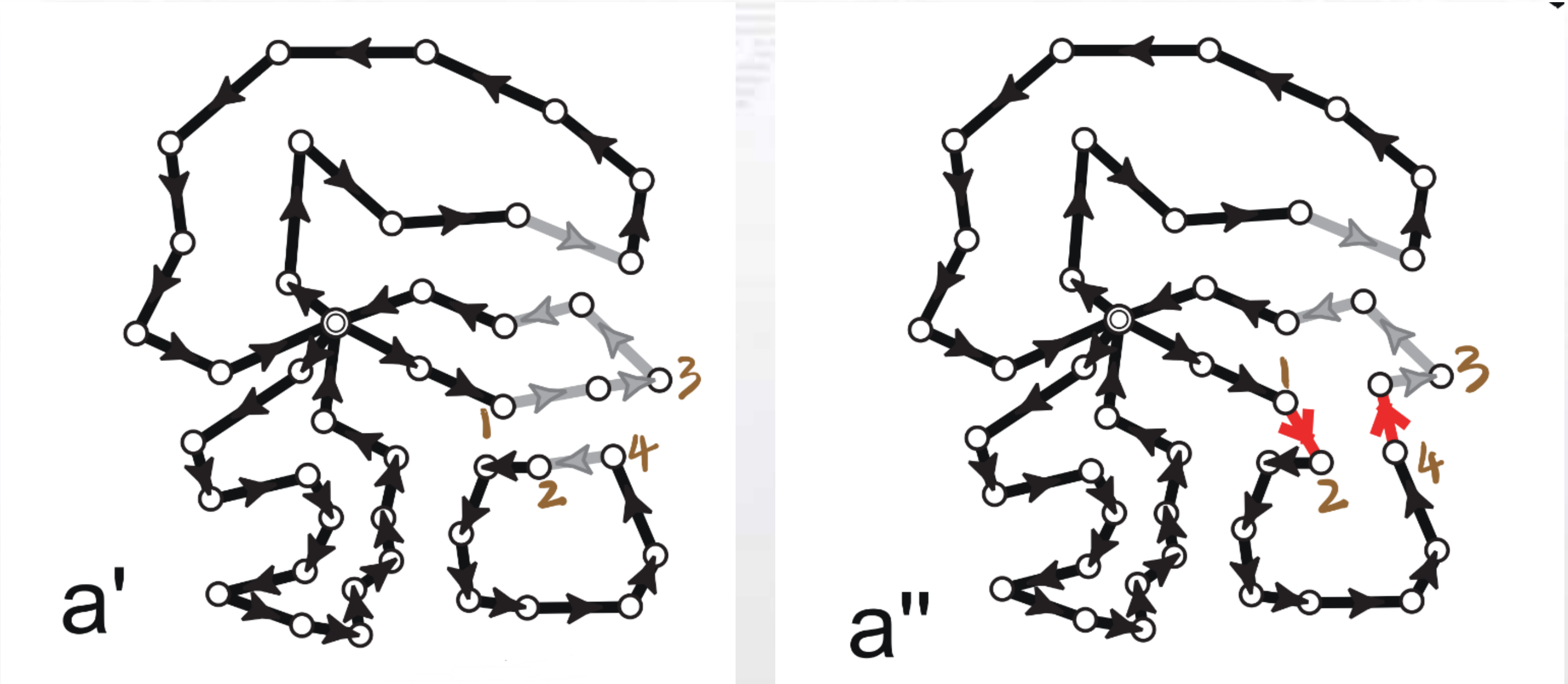
4.3

交叉算子





4.3 交叉算子



4.4 修复程序

4.4 repair

1.随机选取一个违反约束的路径（容量、时间）
2.用该路径上的点构造邻域
3.贪心仅选取可以减小乘法的邻域
4.直到局本部最优或者解合法
5.如果得不到合法解，就丢弃这个解

4.4 repair

```
Procedure REPAIR (σ)
begin
  repeat
    Randomly select an infeasible route r;
    Search σ′ ∈ ⋃_{v∈cust(r)} N(v,σ) that minimizes F_g(σ′) subject to α·ΔP_c + β·ΔP_tw < 0;
    if σ′ exists then Update σ := σ′;
    else break ;
  until σ becomes an feasible solution
  return σ;
end
```

Fig. 5. Algorithm of the repair procedure.

1.随机选取一个违反约束的路径（容量、时间）
2.用该路径上的点构造邻域
3.贪心仅选取可以减小乘法的邻域
4.直到局本部最优或者解合法
5.如果得不到合法解，就丢弃这个解

4.4

限制邻域动作

- 去掉in-relocate动作
  - 因为加一个节点进一个非法的路径
  - 既不能减小容量惩罚也不能减小时间窗的惩罚
- 每次选取下降最快的动作
  - 可以加速迭代
  - 同时尽量保存继承来的特征
- 优先选取违反时间窗的路径

4.4

动作评估

容量惩罚变化量计算可以在O(1)内解决
时间窗惩罚的变化评估大多数也可在O(1)解决
有些情况依然需要O(n)
在附录提到的方法（4.6)

4.5 局部搜索

4.5 局部搜索

Procedure Local _Search (σ)
begin
1 : V_{new} is initialized with a set of customers in the new routes not existing in p_A;
2 : **repeat**
3 : Randomly select $\sigma' \in \bigcup_{v \in V new} N(v,\sigma)$ such that $F(\sigma') < F(\sigma)$;
4 : **if** σ' exists **then** Update $\sigma := \sigma'$;
5 : **until** no improvement move is found
6 : **return** σ ;
end

Fig. 6. Algorithm of the local search.

4.5

限制邻域

用Vnew中的点来构造邻域

Vnew中不包含Pa中的点

更容易找到改进的动作

因为Pa 之前已经经过局部搜索

不在Vnew中的点也可能被选中

因为邻域动作还需要选择点w

4.6 Ptw快速评估

4.6 av

$$\tilde{a}_{v_0} = e_0, (\tilde{a}'_{v_0} = e_0),$$

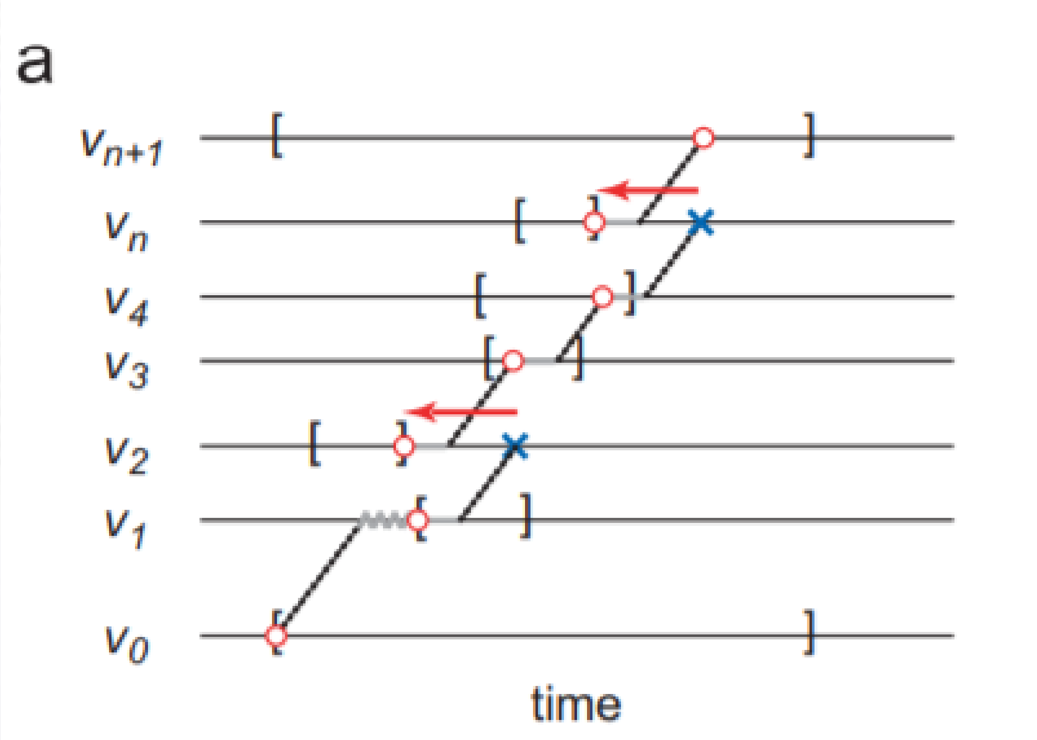
$$\tilde{a}'_{v_i} = \tilde{a}_{v_{i-1}} + s_{v_{i-1}} + c_{v_{i-1}v_i} \quad (i = 1, \dots, n+1),$$

$$\begin{cases} \tilde{a}_{v_i} = \max\{\tilde{a}'_{v_i}, e_{v_i}\} & \text{if } \tilde{a}'_{v_i} \le l_{v_i} \\ \tilde{a}_{v_i} = l_{v_i} & \text{if } \tilde{a}'_{v_i} > l_{v_i} \end{cases} \quad (i = 1, \dots, n+1). \tag{4}$$

$$TW^{\rightarrow}_{v_i} = \sum_{j=0}^{i} \max\{\tilde{a}'_{v_j} - l_{v_j}, 0\} \quad (i = 0, \dots, n+1). \tag{6}$$

4.6

av



4.6 Zv

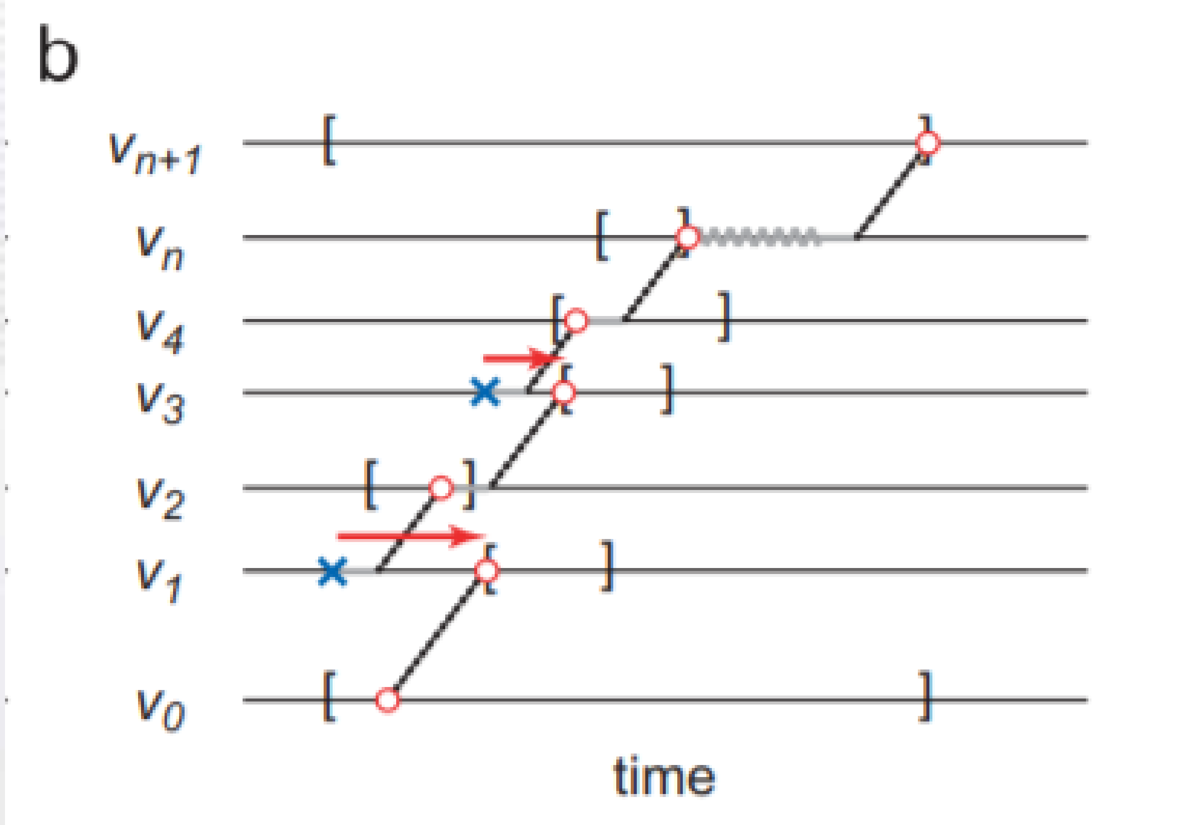
$$\tilde{z}_{v_{n+1}} = l_0 \quad (\tilde{z}'_{v_{n+1}} = l_0),$$

$$\tilde{z}'_{v_i} = \tilde{z}_{v_{i+1}} - c_{v_i v_{i+1}} - s_{v_i} \quad (i = 0, \ldots, n),$$

$$\begin{cases} \tilde{z}_{v_i} = \min\{\tilde{z}'_{v_i}, l_{v_i}\} & \text{if } \tilde{z}'_{v_i} \geq e_{v_i} \\ \tilde{z}_{v_i} = e_{v_i} & \text{if } \tilde{z}'_{v_i} < e_{v_i} \end{cases} \quad (i = 0, \ldots, n), \tag{7}$$

$$TW^{\leftarrow}_{v_i} = \sum_{j=i}^{n+1} \max\{e_{v_j} - \tilde{z}'_{v_j}, 0\} \quad (i = 0, \ldots, n+1). \tag{8}$$

4.6 Zv



4.6

Ptw切分

$$TW(r) = TW^{\rightarrow}_{v_i} + TW^{\leftarrow}_{v_i} + \max\{\tilde{a}_{v_i} - \tilde{z}_{v_i}, 0\} \quad (i = 0, \dots, n+1), \tag{9a}$$

$$TW(r) = TW^{\rightarrow}_{v_{i-1}} + TW^{\leftarrow}_{v_i} + \max\{\tilde{a}'_{v_i} - \tilde{z}_{v_i}, 0\} \quad (i = 1, \dots, n+1), \tag{9b}$$

$$TW(r) = TW^{\rightarrow}_{v_{i-1}} + TW^{\leftarrow}_{v_{i+1}} + \max\{\tilde{a}'_{v_i} - \tilde{z}'_{v_i}, 0\} \quad (i = 1, \dots, n). \tag{9c}$$

4.6 合并的代价

$$\max\{\tilde{a}_{v_i} - \tilde{z}_{v_i}, 0\}$$

$$\max\{\tilde{a}'_{v_i} - \tilde{z}_{v_i}, 0\}$$

$$\max\{\tilde{a}'_{v_i} - \tilde{z}'_{v_i}, 0\}$$

在Vi点计算用av 和 Zv评估时间窗惩罚

这时候会出现 av>Zv的情况

还需要额外的av-Zv的惩罚

4.6

命题1

Proposition 1. *Let $\tilde{a}_v$, $\tilde{z}_v$, $TW_v^{\rightarrow}$ and $TW_v^{\leftarrow}$ ($v \in V$) be known for a current solutionσ.*

If a route $\langle 0, \dots, x, v, \dots, 0\rangle$ is generated from two partial paths $\langle 0, \dots, x\rangle$ and $\langle v, \dots, 0\rangle$, the time window penalty of this route is computed by

$$TW_x^{\rightarrow} + TW_v^{\leftarrow} + \max\{\tilde{a}_x + s_x + c_{xv} - \tilde{z}_v, 0\}.$$

If a route $\langle 0, \dots, x, v, y, \dots, 0\rangle$ is generated from two partial paths $\langle 0, \dots, x\rangle$, $\langle y, \dots, 0\rangle$ and customer v, the time window penalty of this route is computed by

$$TW_x^{\rightarrow} + TW_y^{\leftarrow} + \max\{\tilde{a}_x + s_x + c_{xv} - (\tilde{z}_y - c_{vy} - s_v), 0\}.$$

O(1)评估

4.6

例外

However, a randomly selected intra-route move from Out-Relocate(v, σ) and Exchange(v, σ)[4] (2-opt* is not defined in this situation) cannot be computed in constant time. In these cases, the worst case computation time is $O(n)$. Every time solution σ is updated, $\tilde{a}_v$, $\tilde{z}_v$, $TW_v^{\rightarrow}$ and $TW_v^{\leftarrow}$ ($v \in$ updated routes) must be recalculated and it takes $O(n)$ time. The actual computation times for these updates are negligible because the number of updates is much lower than the number of evaluations for the moves in the repair procedure.

①在同一条路径内的结点交换和结点插入

②每次更新都要重新计算a ,Z, TW->TW<-，需要花费O(n)

4.6 命题2

Proposition 2. *For a given route* $r = \langle 0, \ldots, v^-, v, v^+, \ldots, 0 \rangle$, *if* $TW^{\rightarrow}_{v^-} + TW^{\leftarrow}_{v^+} = TW(r)$, *any move from* $\mathcal{N}(v, \sigma)$ *except for intra-route exchange does not decrease the time window penalty originating from this route.*

如果V不影响r的时间窗惩罚
那么只有在一条路径内的exchange可以改善惩罚

4.6 命题2-证明

Proposition 2 is proven as follows. If the above condition holds and customer v is removed from route r, we can see that $TW(r)$ does not change. Thus, $TW(r)$ (and $P_{tw}(\sigma)$) is never decreased by a move from Out-Relocate(v, σ). In the same way, $TW(r)$ is never decreased by an inter-route move from Exchange(v, σ) (the time window penalty of the other route may be decreased).

拿走v不能降低Ptw
加入一个点更不能降低Ptw

4.6 命题2-证明

new route $\langle 0,\ldots,v^-,w^+,\ldots,0\rangle$ and $\langle 0,\ldots,w,v,\ldots,0\rangle$ are generated by a move from 2-opt*(v,σ), we can see that the two partial paths $\langle 0,\ldots,v^-\rangle$ and $\langle v,\ldots,0\rangle$, which originally form route r, bring down the penalties $TW^{\rightarrow}_{v^-}$ and $TW^{\leftarrow}_{v}$ in the new two routes. Here, $TW^{\leftarrow}_{v}$ is equal to $TW^{\leftarrow}_{v^+}$ if the assumption is met because, in general, $TW^{\rightarrow}_{v^-} + TW^{\leftarrow}_{v} \leq TW(r)$ and $TW^{\leftarrow}_{v} \geq TW^{\leftarrow}_{v^+}$ hold. Only intra-route moves from Exchange(v,σ) may decrease $TW(r)$ even if the assumption is met because w^+ (or w^-) is replaced with v in the same route.

在v点处 Zv’ 落在了v的时间窗内

$$TW^{\rightarrow}_{x} + TW^{\leftarrow}_{v} + \max\{\tilde{a}_x + s_x + c_{xv} - \tilde{z}_v, 0\}.$$

Part 05

总结

5.1 summary

交叉算子设计：
继承父代的优点，与父代不一样

快速评估的方法：
可以大大减少运行时间

剪枝：
缩小搜索解空间的范围，提高速度

谢谢观赏

参考文献

[1-2] https://developers.google.cn/optimization/routing/vrp
[3] https://www.cnblogs.com/osmondwang/p/7244546.html
[4]https://zhuanlan.zhihu.com/p/62516988
[5]Yuichi Nagata and Olli Braysy.
A powerful route minimization heuristic for the
vehicle routing problem with time windows.
Operations Research Letters, 37(5):333–338, 2009.
[6]Yuichi Nagata, Olli Braysy, and Wout Dullaert.
A penalty based edge assembly memetic algorithm for the
vehicle routing problem with time windows.
Computers & operations research, 37(4):724–737, 2010.
[7] The Vehicle Routing Problem with Time Windows Guy
Desaulniers Oli B.G. Madsen Stefan Ropke