交通工程学
Introduction to Traffic Engineering
第 15 节　车辆路径问题[1]

葛乾

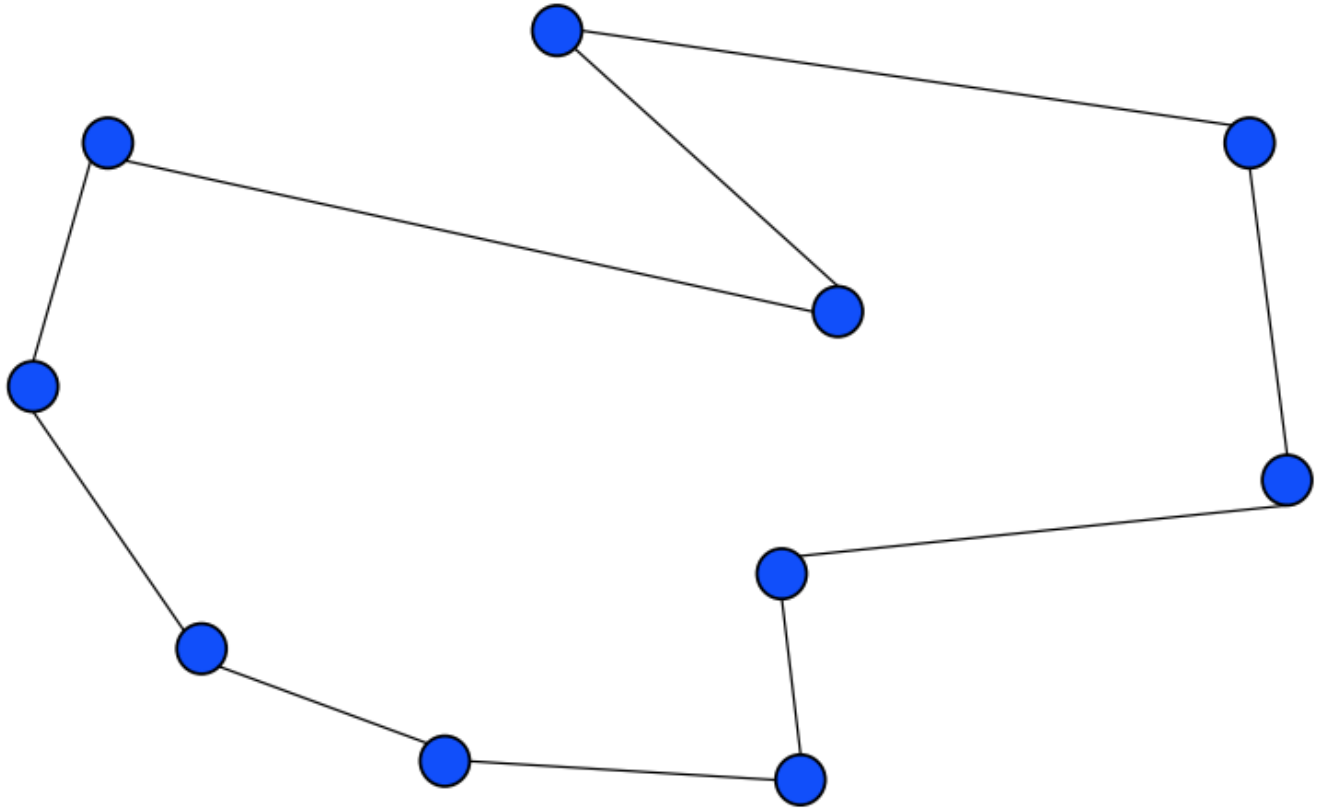
西南交通大学　系统科学与系统工程研究所
西南交通大学　交通工程系

[1]基于乔治亚理工大学 Martin Savelsbergh 教授讲义。仅供教学使用，请勿传播。

本节目录

旅行商问题回顾

- In the TSP the objective is to find the shortest tour through a set of cities, visiting each city exactly once and returning to the starting city.
- Type of decisions:
  - routing

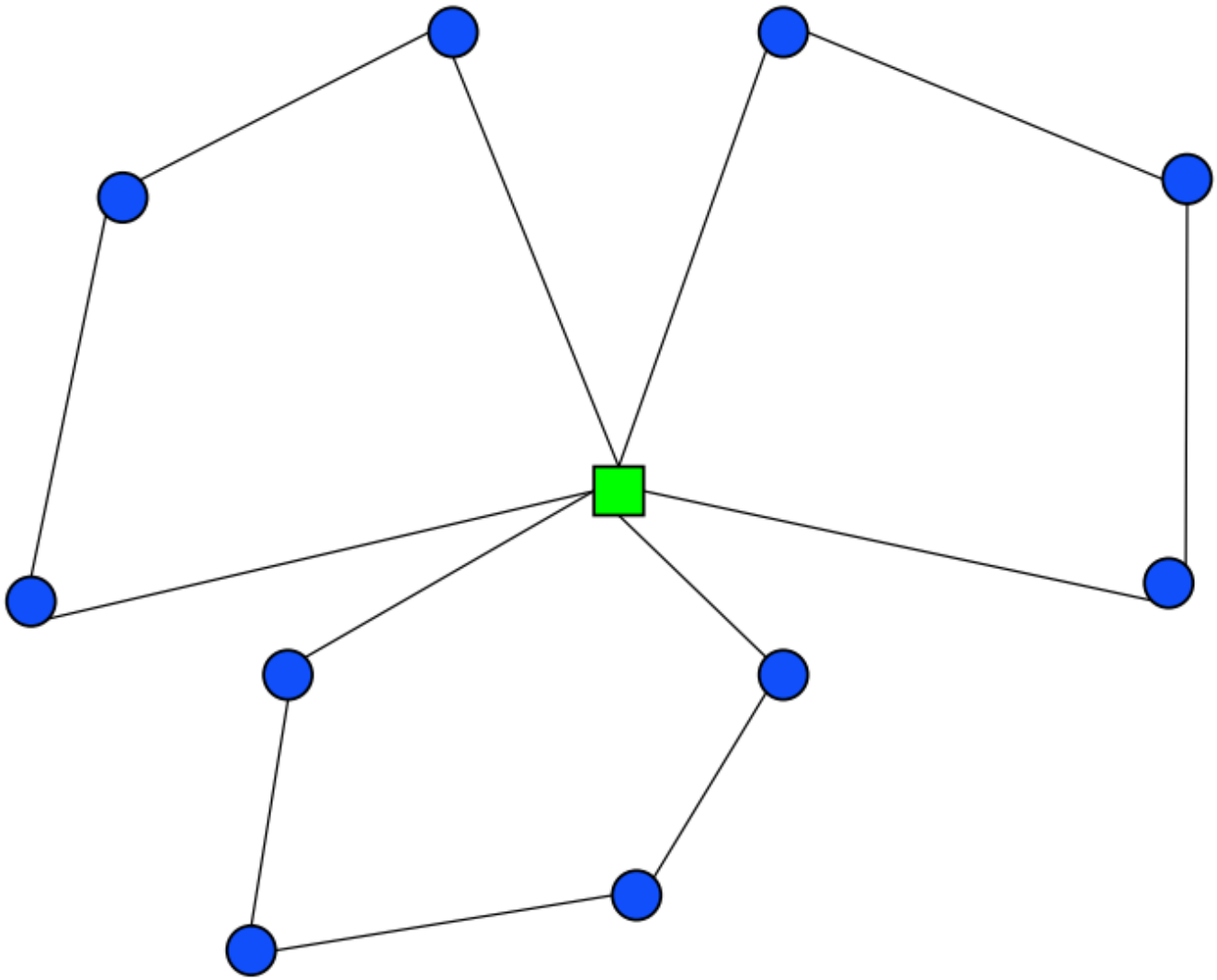


Figure

Vehicle Routing Problem

- In the VRP a number of vehicles located at a central depot has to serve a set of geographically dispersed customers. Each vehicle has a given capacity and each customer has a given demand. The objective is to minimize the total distance traveled.
- Type of decisions
  - assigning
  - routing

Vehicle Routing Problem



Figure

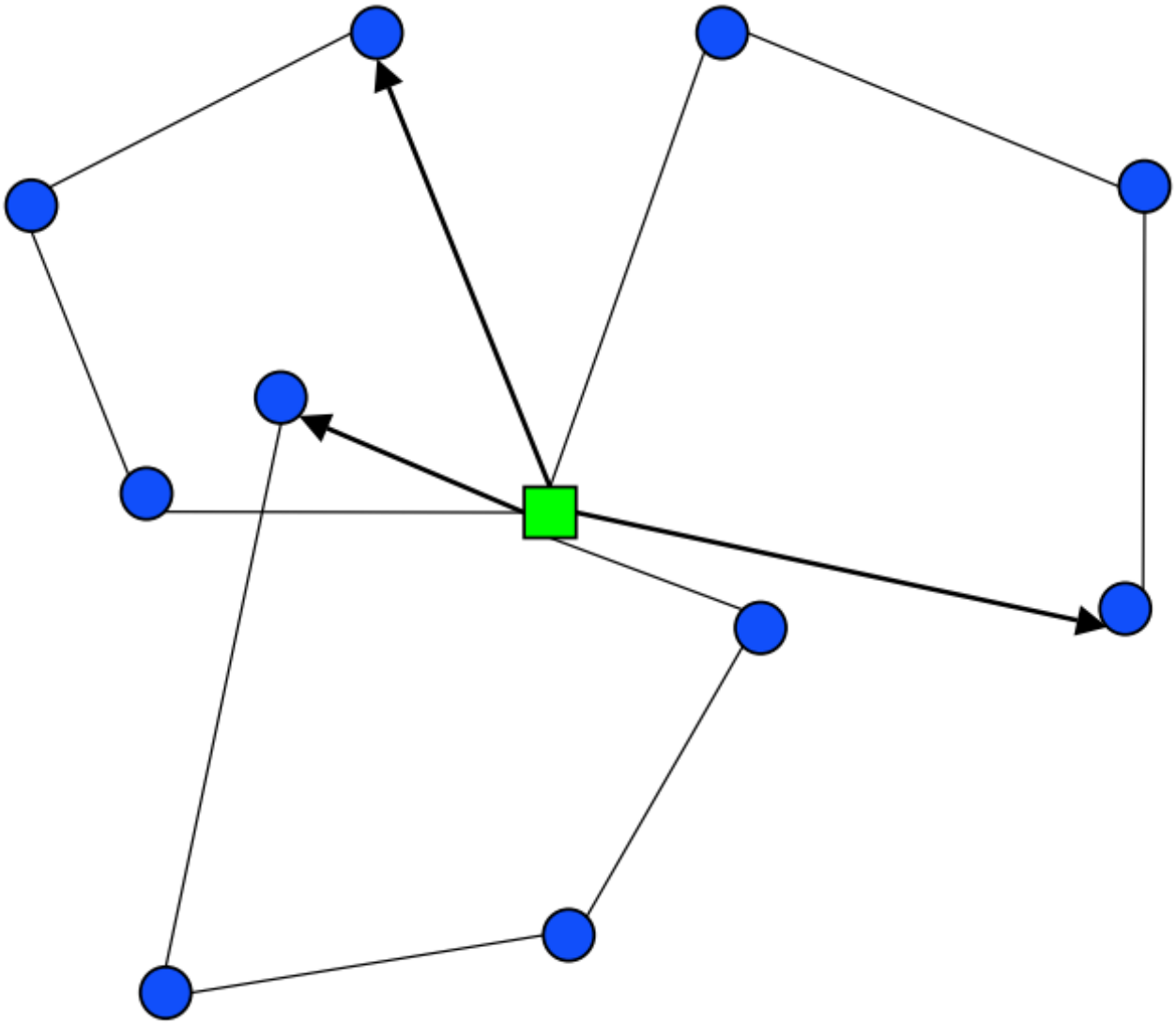
与 TSP 的关系

- TSP 可视为 VRP 的一种特例，单辆车，无容量与需求限制...
- 因此，用于求解 TSP 的启发式规则经过略作修改也可用于求解 VRP
  - 思考一下：之前用于求解 TSP 的方法（例如插入算法），如何用于 VRP

Vehicle Routing Problem with Time Windows

- In the VRPTW a number of vehicles is located at a central depot and has to serve a set of geographically dispersed customers. Each vehicle has a given capacity. Each customer has a given demand and has to be served within a given time window.
- Type of decisions
  - assigning
  - routing
  - scheduling

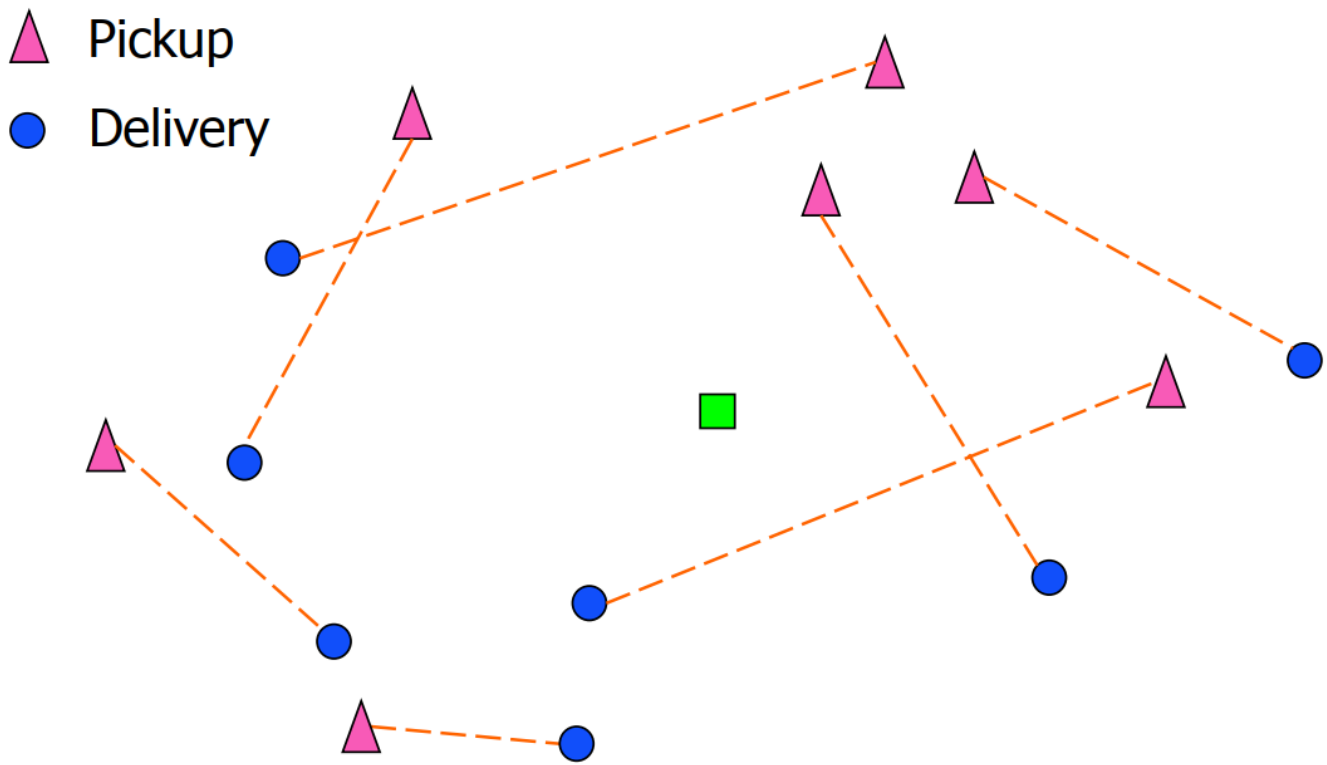
VRPTW



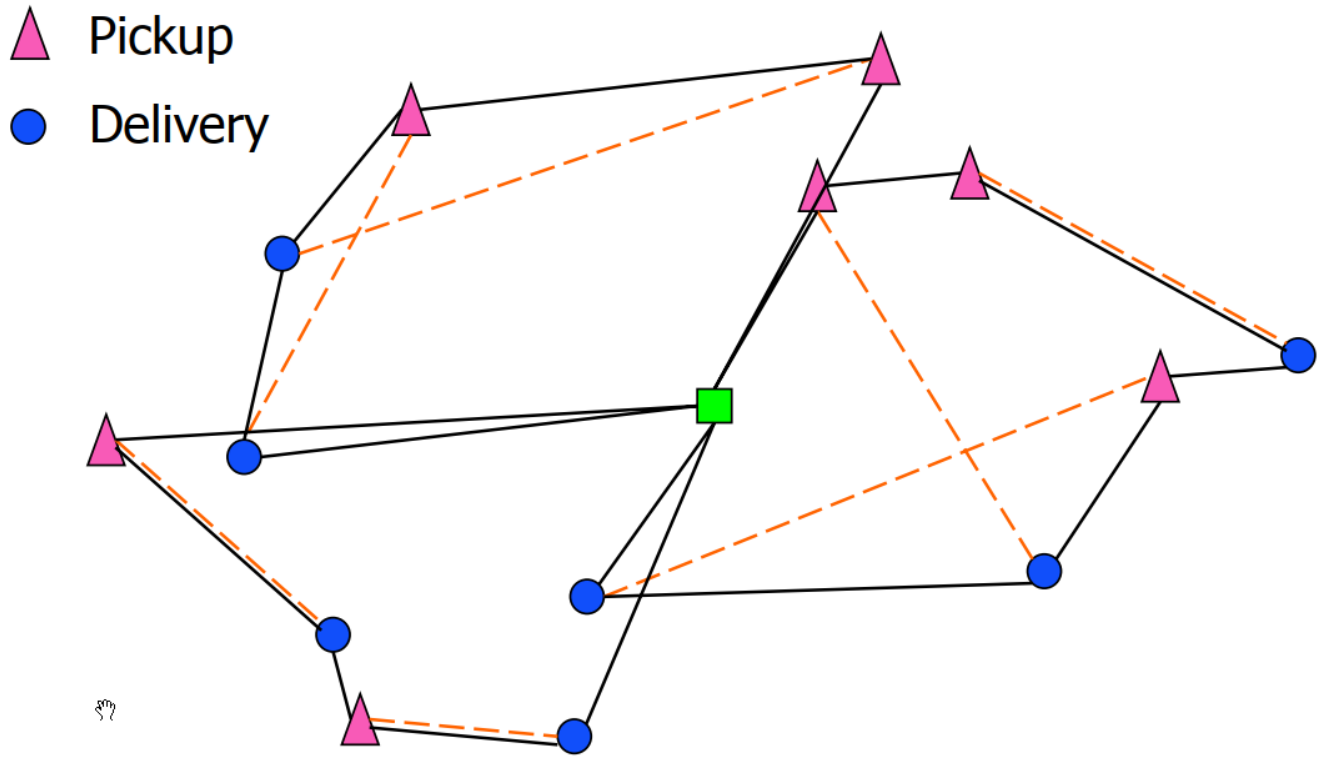
Pickup and Delivery Problem with Time Windows

- In the PDPTW a number of vehicles has to serve a number of transportation requests. Each vehicle has a given capacity. Each transportation request specifies the size of the load to be transported, the location where it is to be picked up plus a pickup time window, and the location where it is to be delivered plus a delivery time window.
- Type of decisions
  - assigning
  - routing
  - scheduling

PDPTW



PDPTW



路径与调度问题

- 优化目标
  - minimize vehicles
  - minimize miles
  - minimize labor
  - satisfy service requirements
  - maximize orders
  - maximize volume delivered per mile

路径与调度问题

- 优化目标
  - minimize vehicles
  - minimize miles
  - minimize labor
  - satisfy service requirements
  - maximize orders
  - maximize volume delivered per mile

路径与调度问题

- 现实考虑
  - Single vs. multiple depots
  - Vehicle capacity: homogenous vs. heterogeneous; volume vs. weight
  - Driver availability: fixed vs. variable start times
  - Delivery windows: hard vs. soft; single vs. multiple; periodic schedules
  - Service requirements: Maximum ride time; Maximum wait time
  - Fixed and variable delivery times
  - Fixed vs. variable regions/route

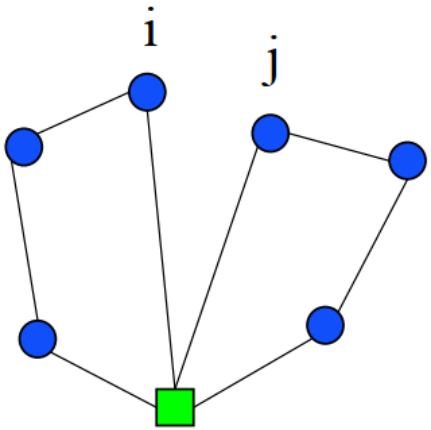
新的变种

- Dynamic routing and scheduling problems
  - More and more important due to availability of GPS and wireless communication
  - Information available to design a set of routes and schedules is revealed dynamically to the decision maker: Order information (e.g., pickups), Vehicle status information (e.g., delays), Exception handling (e.g., vehicle breakdown)
- Stochastic routing and scheduling problems
  - Size of demand
  - Travel times

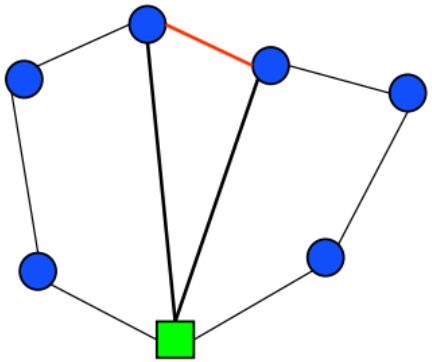
本节目录

本节目录

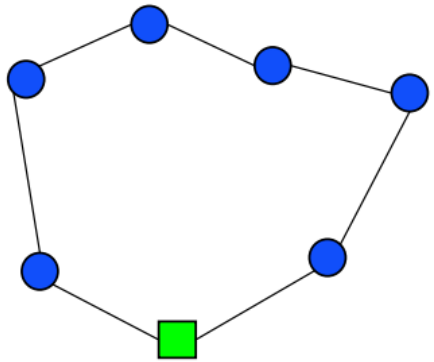
节省费用



Initial



Intermediate



Final

节省费用为 $s(i,j) = c(0,i) + c(0,j) - c(i,j)$

Clarke-Wright 算法

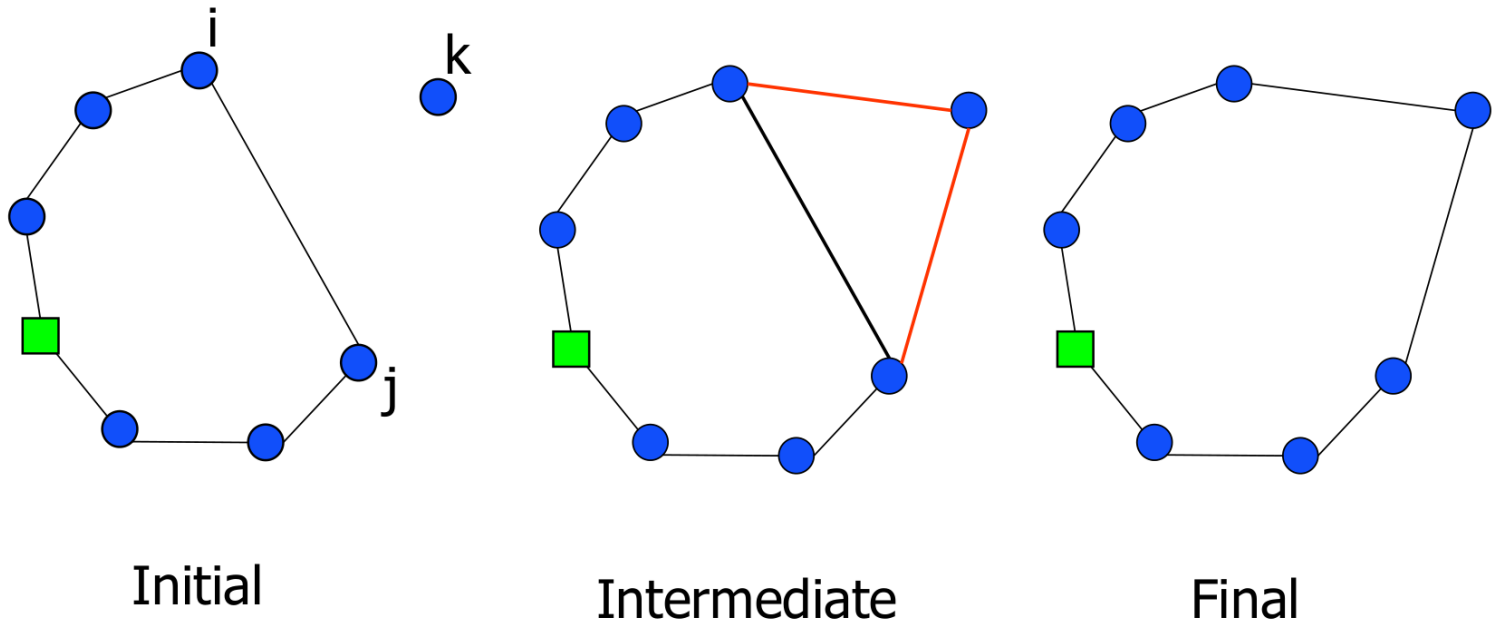
1. Make n routes: $v_0 \to v_i \to v_0$, for each $i \geqslant 1$;
2. Compute the savings for merging delivery locations i and j for all $i, j \geqslant 1$ and $i \neq j$;
3. Sort the savings in descending order; Create out-and-back routes for all customers.
4. Starting at the top of the (remaining) list of savings, merge the two routes associated with the largest (remaining) savings, provided that:
   - The two delivery locations are not already on the same route;
   - Neither delivery location is interior to its route, meaning that both nodes are still directly connected to the depot on their respective routes;
   - The demand G and distance constraints D are not violated by the merged route.
5. Repeat step 3 until no additional savings can be achieved.

增强策略

- 节省费用为 $s(i,j) = c(0,i) + c(0,j) - \lambda c(i,j)$
- The larger λ, the more emphasis is placed on the distance between customers being connected

本节目录

插入算法



Figure

算法

1. Start with a set of unrouted stops;
2. Select an unrouted stop; Insert selected stop in current set of routes;
3. Repeat step 2 until no unrouted stop is available.

类似 TSP 问题，可通过不同准则确定选择和插入的点：最近添加，最近插入，最远插入，最省插入

Nearest addition

- Selection: If partial tour T does not include all cities, find cities k and j, j on the tour and k not, for which $c(j,k)$ is minimized.
- Insertion: Let $\{i,j\}$ be either one of the two edges involving j in T, and replace it by $\{i,k\}$ and $\{k,j\}$ to obtain a new tour including k.

Nearest Insertion

- Selection: If partial tour T does not include all cities, find cities k and j, j on the tour and k not, for which $c(j,k)$ is minimized.
- Insertion: Let $\{i,j\}$ be the edge of T which minimizes $c(i,k)+c(k,j)-c(i,j)$, and replace it by $\{i,k\}$ and $\{k,j\}$ to obtain a new tour including k.

Farthest Insertion

- Selection: If partial tour T does not include all cities, find cities k and j, j on the tour and k not, for which $c(j,k)$ is **maximized**.
- Insertion: Let $\{i,j\}$ be the edge of T which minimizes $c(i,k)+c(k,j)-c(i,j)$, and replace it by $\{i,k\}$ and $\{k,j\}$ to obtain a new tour including k.

Cheapest Insertion

- Selection: If partial tour T does not include all cities, find for each k not on T the edge $\{i, j\}$ of T which minimizes $c(T, k) = c(i, k) + c(k, j) - c(i, j)$. Select city k for which $c(T, k)$ is minimized. (**Select the nearest k to any edge in T**)
- Insertion: Let $\{i, j\}$ be the edge of T for which $c(T, k)$ is minimized, and replace it by $\{i, k\}$ and $\{k, j\}$ to obtain a new tour including k.

最坏情况比较

- Nearest addition: 2
- Nearest insertion: 2
- Cheapest insertion: 2
- Farthest insertion: $>$2.43 (Euclidean); $>$6.5 (Triangle inequality)

实施的数据结构

- Priority Queue
  - insert(value, key)
  - getTop(value, key)
  - setTop(value, key)
- k-d Tree
  - deletePt(point)
  - nearest(point)

算法

```
Tree->deletePt(StartPt)
NNOut[StartPt] := Tree->nearest(StartPt)
PQ->insert(Dist(StartPt, NNOut(StartPt)), StartPt)
loop n-1 time
    loop                                          Find nearest point
        PQ->getTop(ThisDist, x)
        y := NNOut[x]
        If y not in tour, then break
                                                  Delayed update
        NNOut[x] = Tree->nearest(x)
        PQ->setTop(Dist(x, NNOut[x]), x)
    Add point y to tour; x is nearest neighbor in tour
                                                  Update
    Tree->deletePt(y)
    NNOut[y] = Tree->nearest(y)
    PQ->insert(Dist(y, NNOut[y]), y)
```

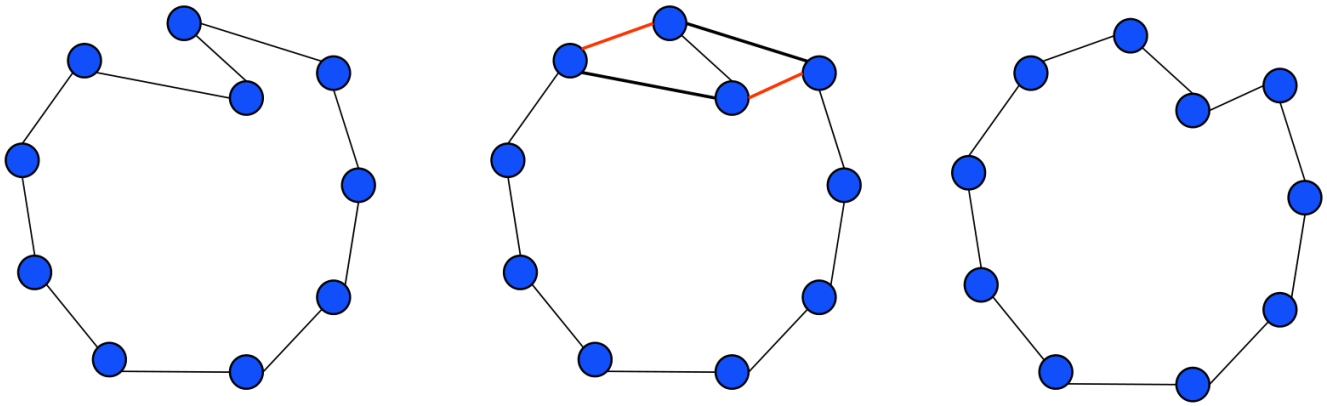
Figure

- Start with a feasible solution x; Define neighborhood $N(x)$.
- Is there an improving neighbor y in $N(x)$ but not in the tour?
  - No. x is locally optimal
  - Otherwise. Add y to tour and start search from y

本节目录

2-change (2-opt)

Take 2 arcs from the tour, reconnect these arcs with each other and calculate new travel distance. If this modification has led to reduction in total travel distance, the current route is updated. Repeat the steps until no more improvements are found or the maximum number of iterations is reached

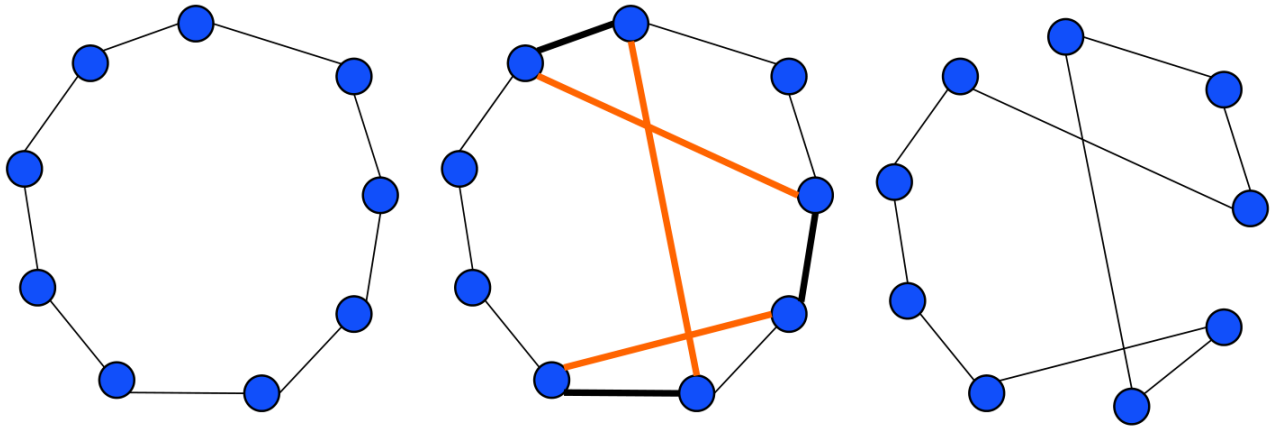


$O(n^2)$ possibilities

Figure

3-change (3-opt)

Similar as the 2-opt except that 3 arcs are taken in each iteration.

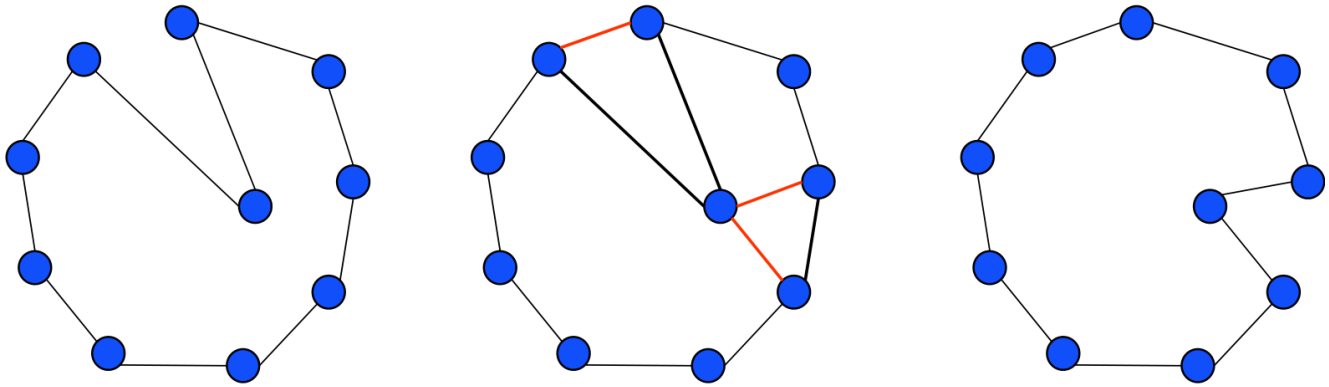


$O(n^3)$ possibilities

Figure

1-relocate

Take one point out and reinsert it in the tour.

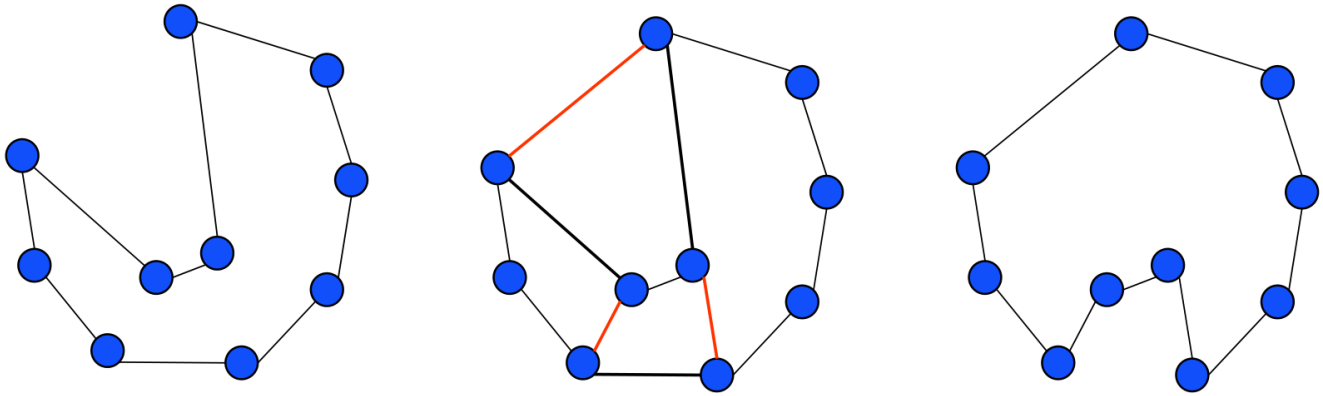


$O(n^2)$ possibilities

Figure

2-relocate

Take two points out and reinsert them in the tour.

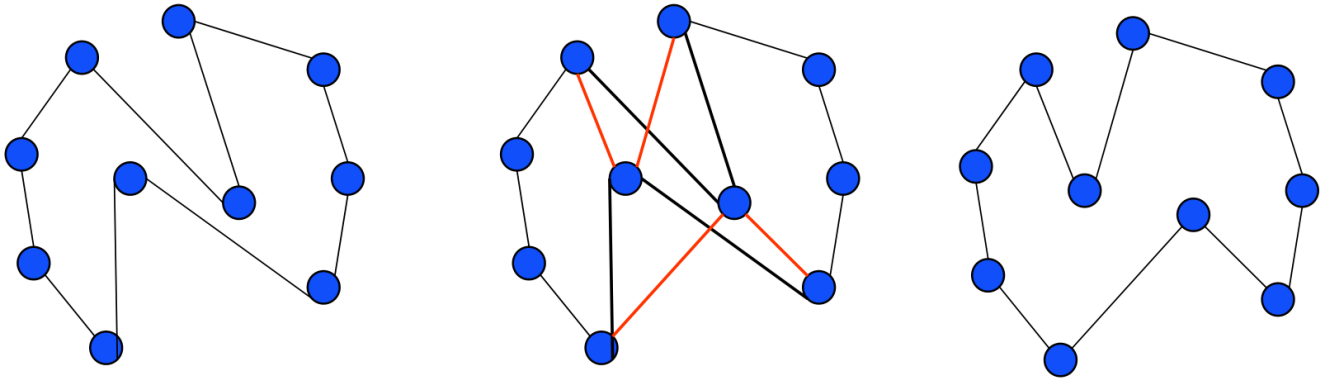


$O(n^2)$ possibilities

Figure

swap

Swap the sequences of two points.

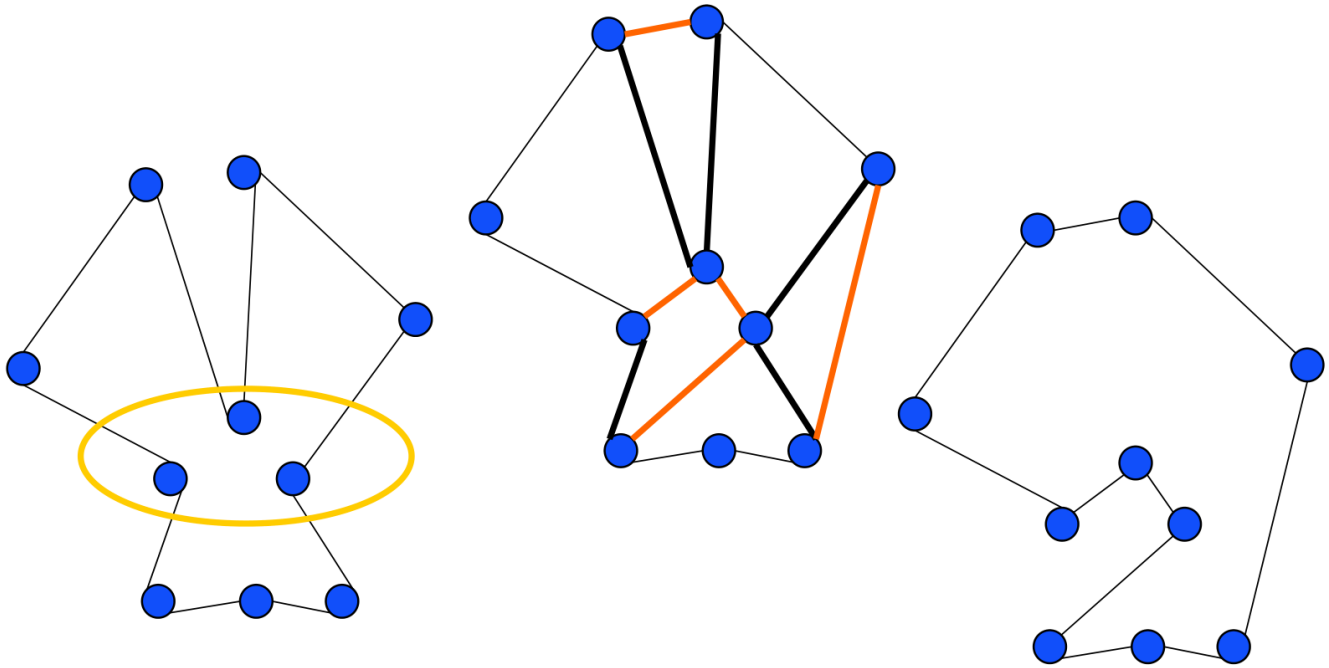


$O(n^2)$ possibilities

Figure

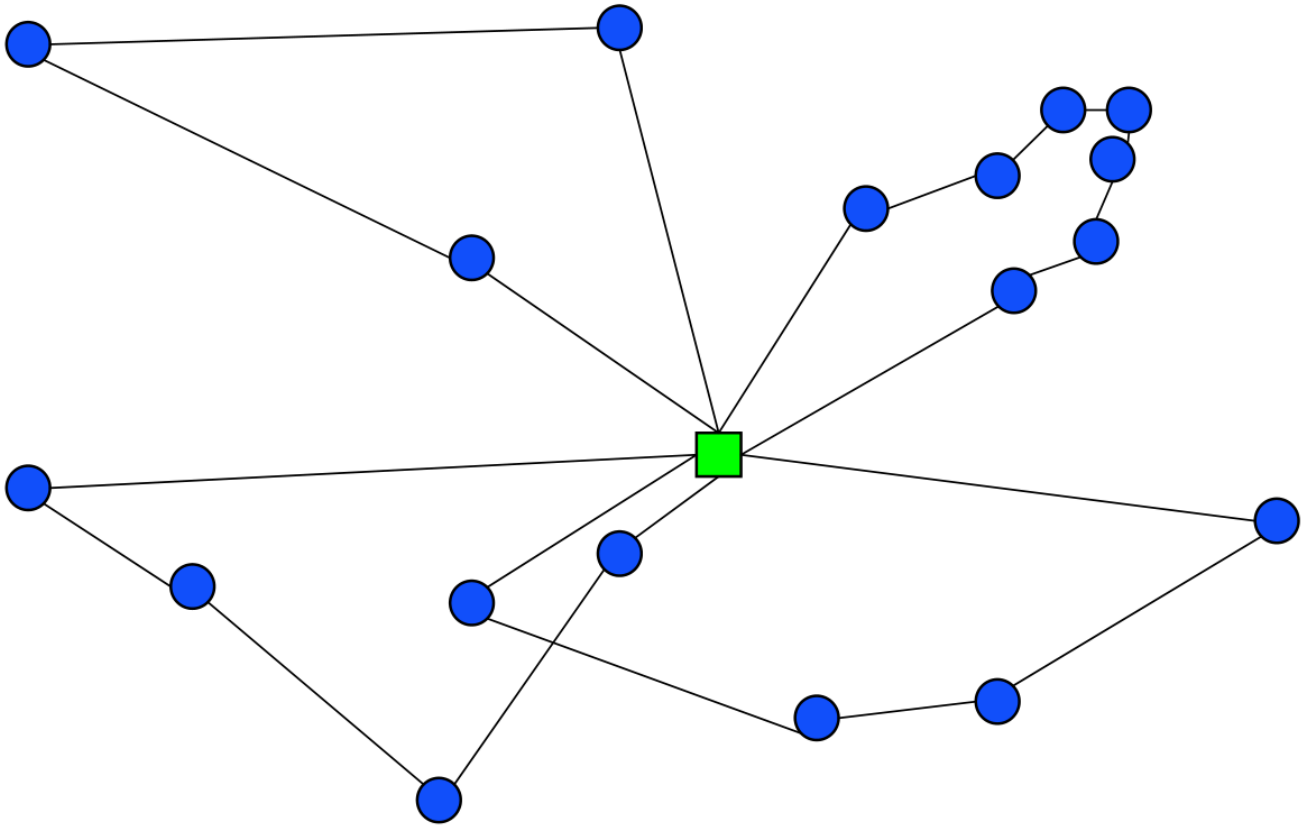
GENI

Select 3 points as the initial subset of tour, and insert the remaining points while conducting 3-opt or k-opt



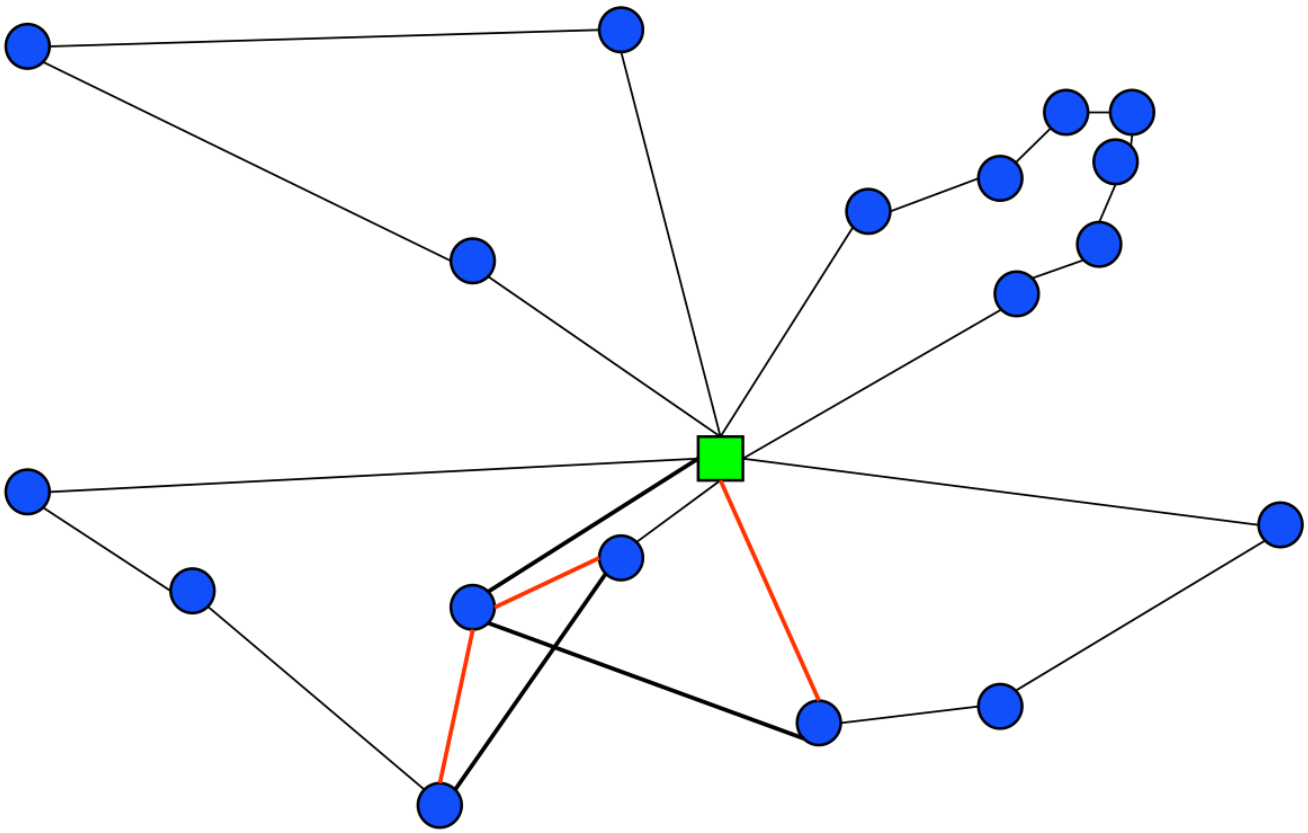
Figure

Vehicle Routing and Scheduling



Figure

Vehicle Routing and Scheduling

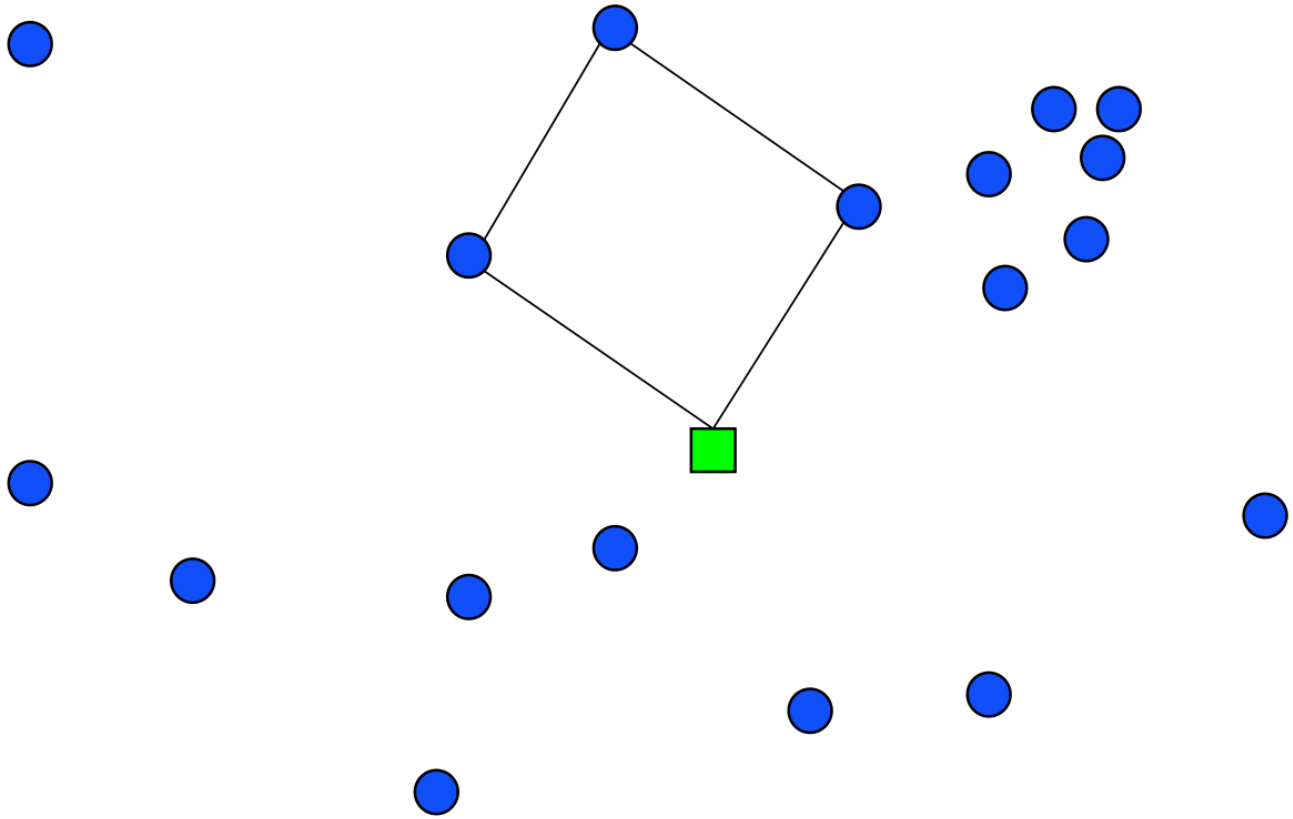


Figure

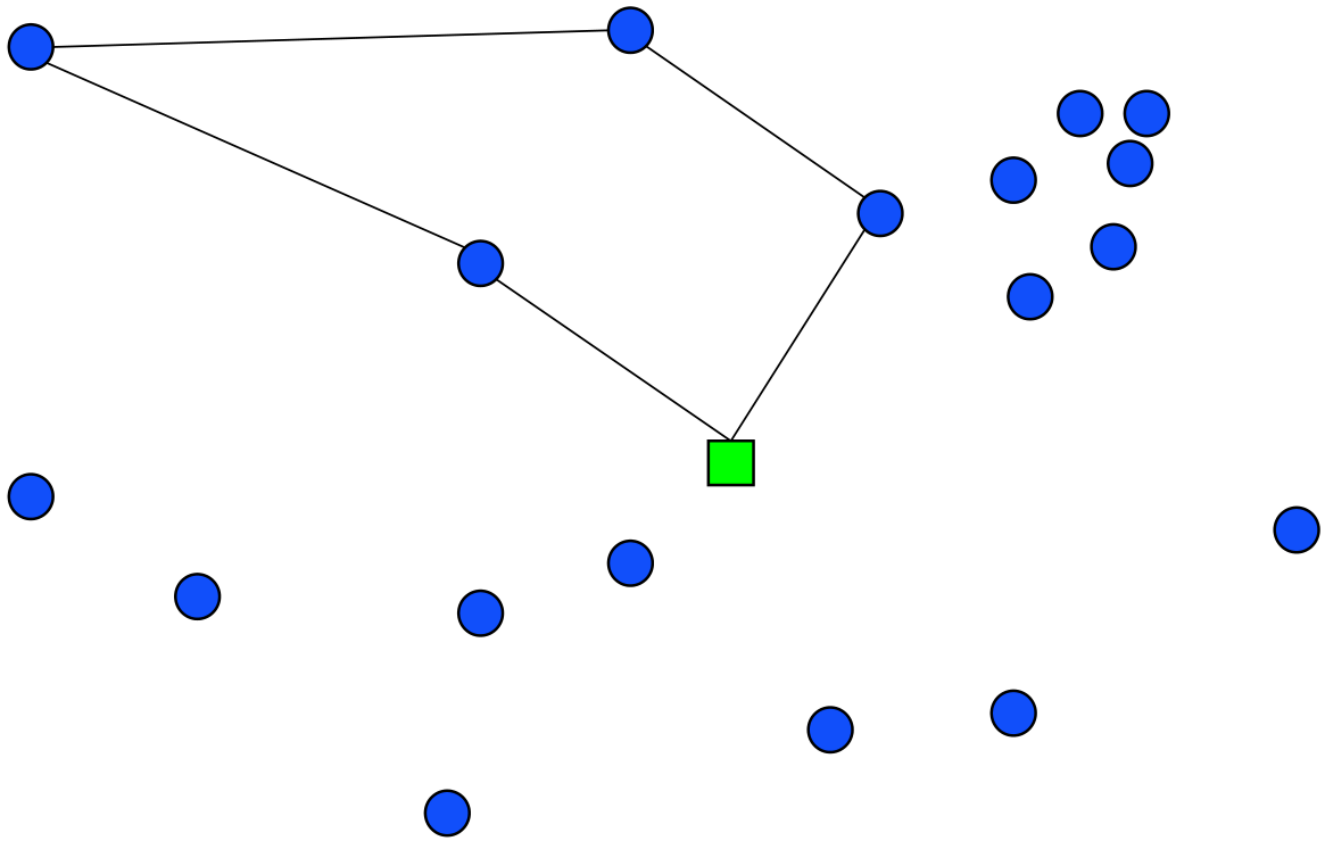
本节目录

基于集合覆盖的建模

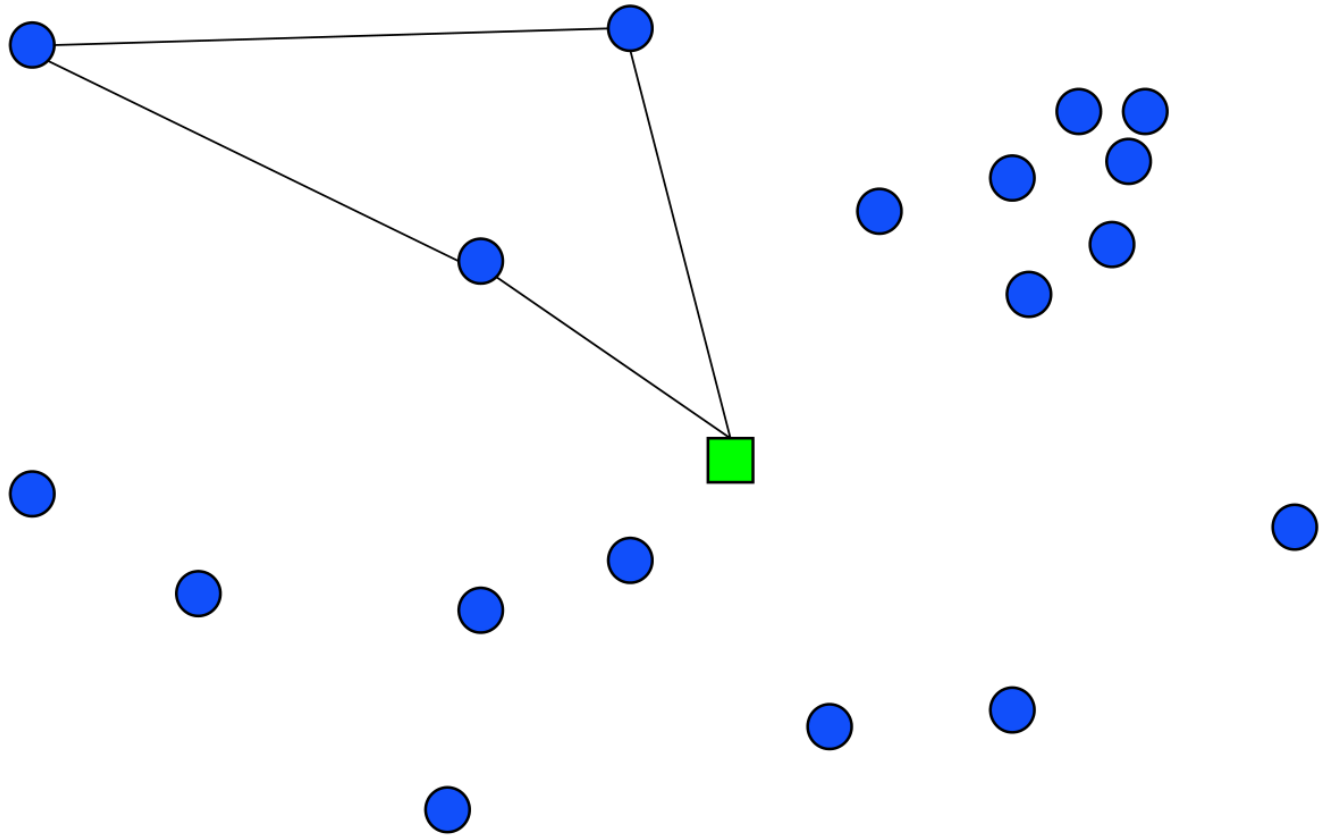
- Assignment decisions are often the most important
- Assignment decisions are often the most difficult



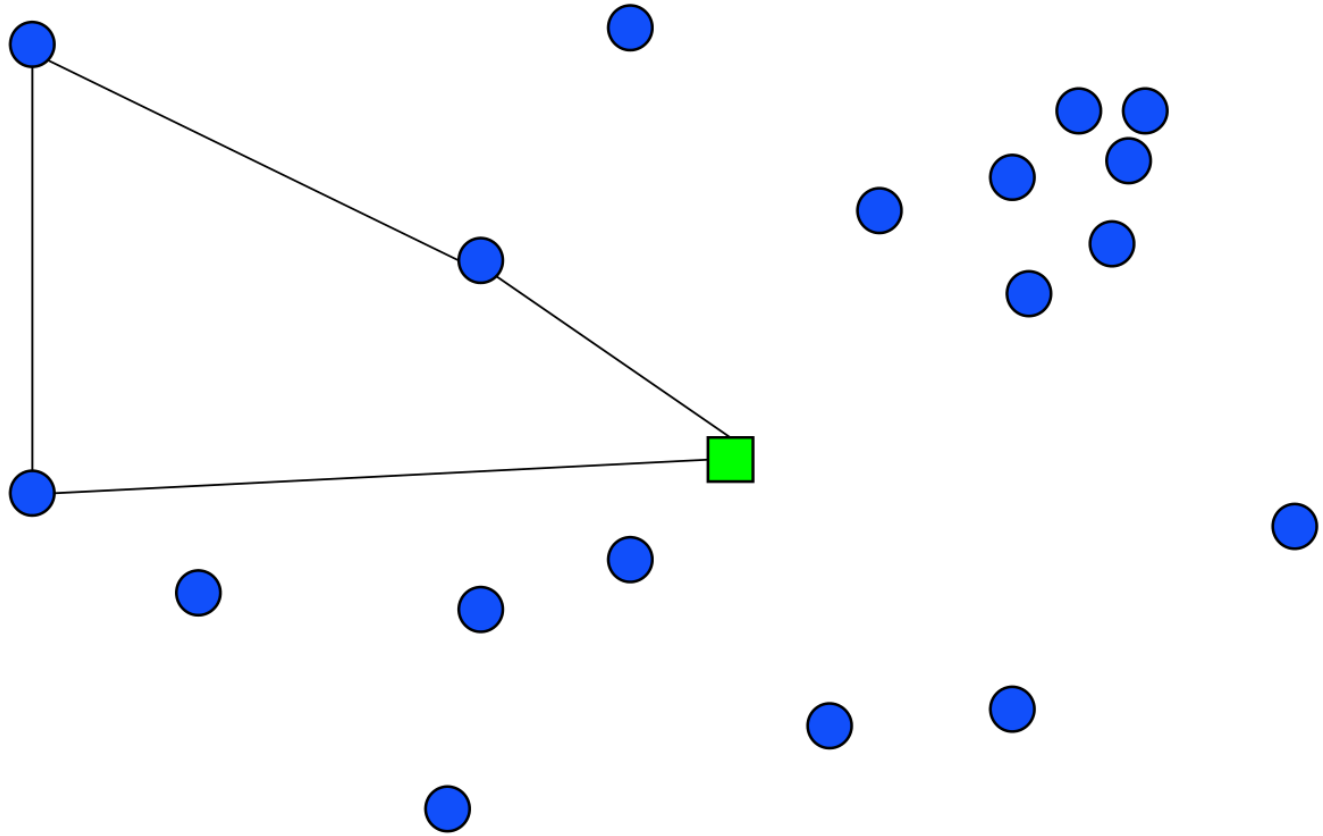
Figure



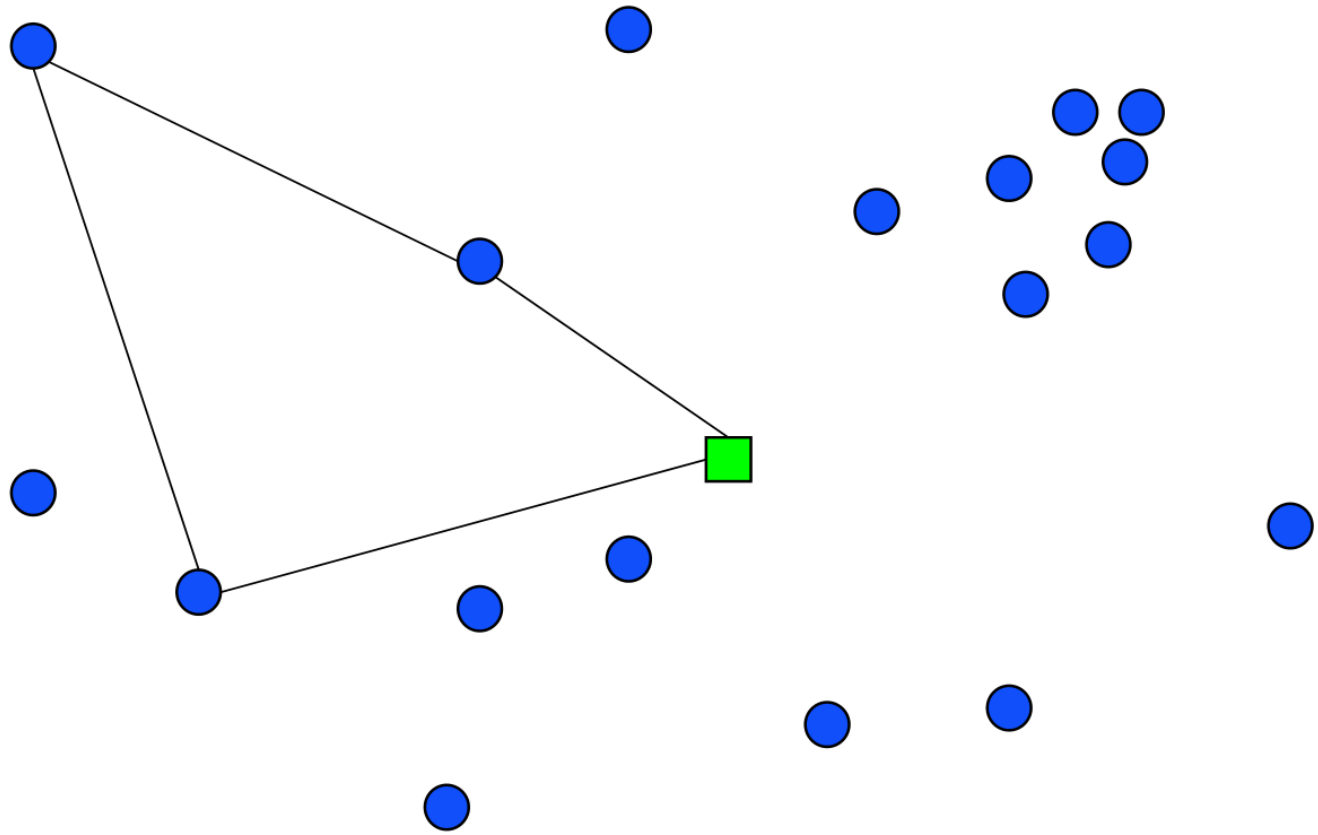
Figure



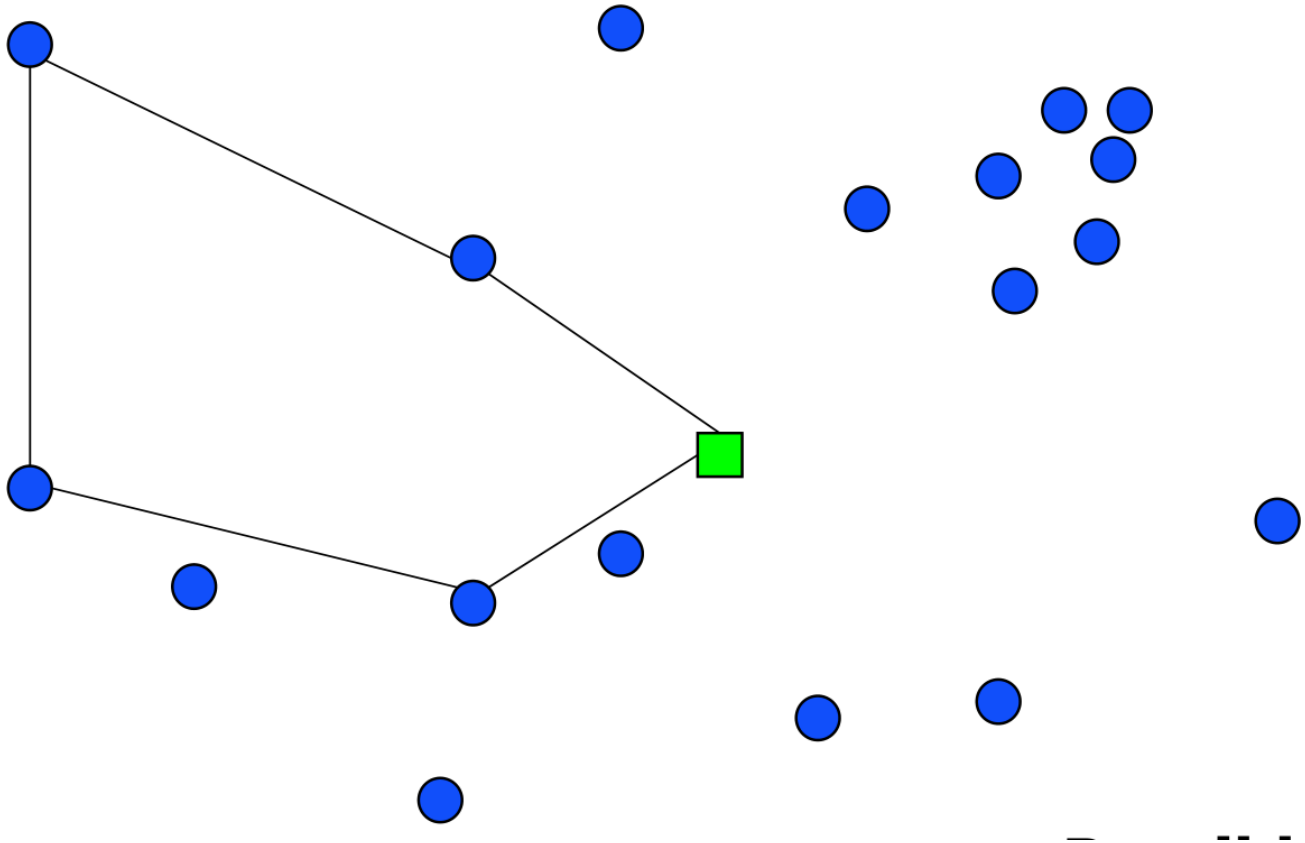
Figure



Figure



Figure



Figure

集合划分问题

$$\min \sum_{k \in K} c_k y_k \tag{1}$$

s.t.

$$\sum_{k \in K} \delta_{kv} y_k = 1 \quad \forall v \in V$$

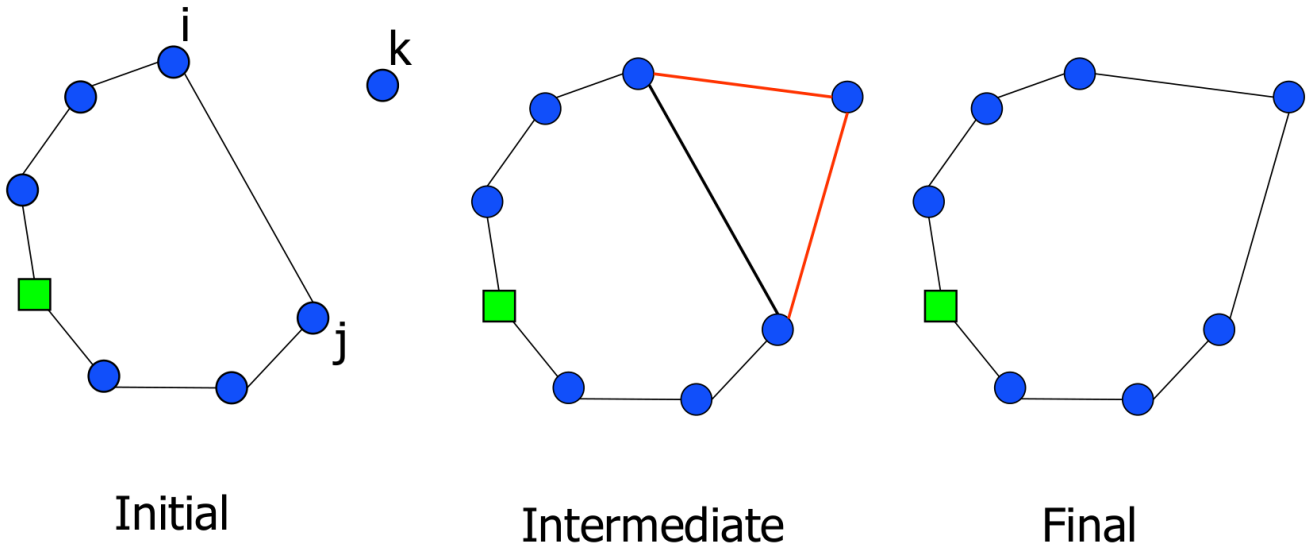
$$y_k \in \{0, 1\}$$

优劣

- Advantage: very flexible
  - heuristics for route generation $\rightarrow$ column generation
  - complicating constraints in route generation
- Disadvantage
  - small to medium size instances

本节目录

本节目录



Figure

优势

- Efficient (fast)
- Effective (good quality solutions)
- Easy to implement
- Easy to extend
  - time windows
  - route duration
  - variable delivery quantities
  - ...

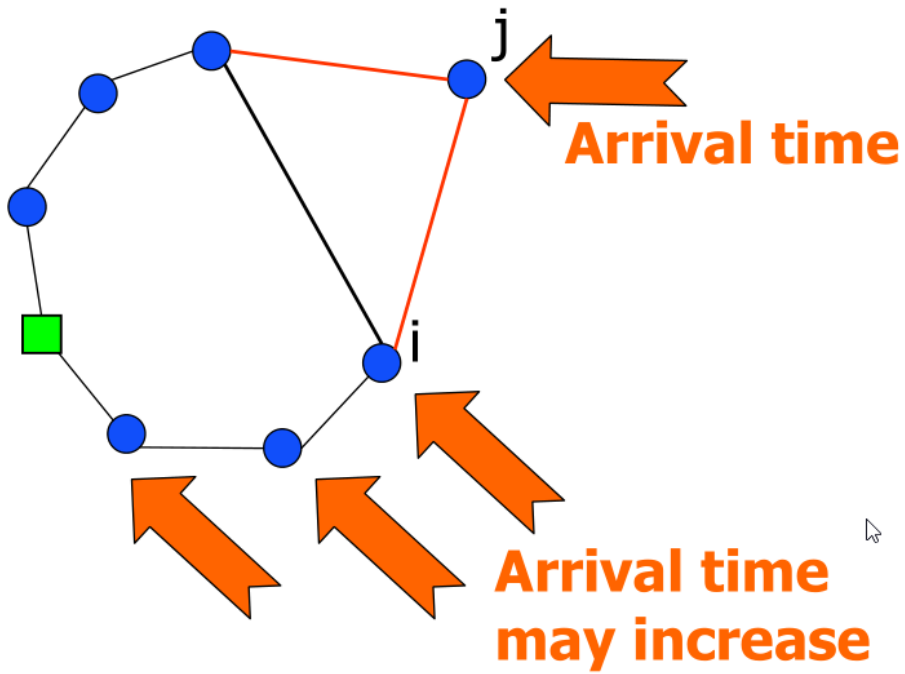
Algorithm 1: Insertion Heuristics

```
1  N ={unassigned customers}
2  R ={set of routes}
3  while N ≠ ∅ do
4  |   p* = −∞
5  |   for j ∈ N do
6  |   |   for r ∈ R and (i − 1, i) ∈ R do
7  |   |   |   if Feasible(i, j) and Profit(i, j)> p* then
8  |   |   |   |   r∗ = r, i∗ = i, j∗ = j, p∗ = Profit(i, j)
9  |   |   |   end
10 |   |   end
11 |   end
12 |   Insert(i*, j*)
13 |   Update(r*)
14 |   N = N\{j*}
15 end
```

算法分析

- Complexity is $O(n^3)$ if Feasible(*), Profit(*), Update(*) are at most $O(n^2)$
- For VRP: Feasible(*): $d_j < Q - q_r$; $q_r = q_r + d_r$

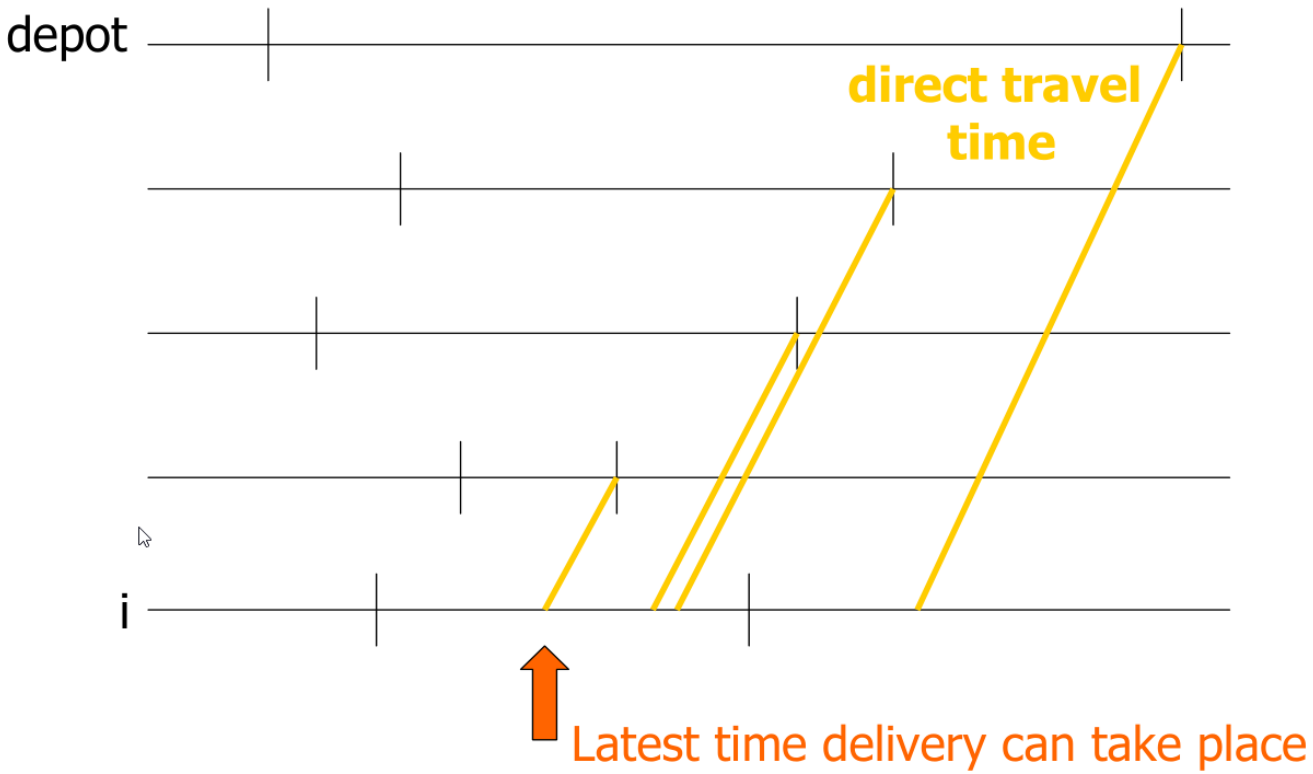
时间窗



Figure

How to quickly check feasibility?

最晚配送时间


depot
direct travel time
i
Latest time delivery can take place

时间窗的可行性

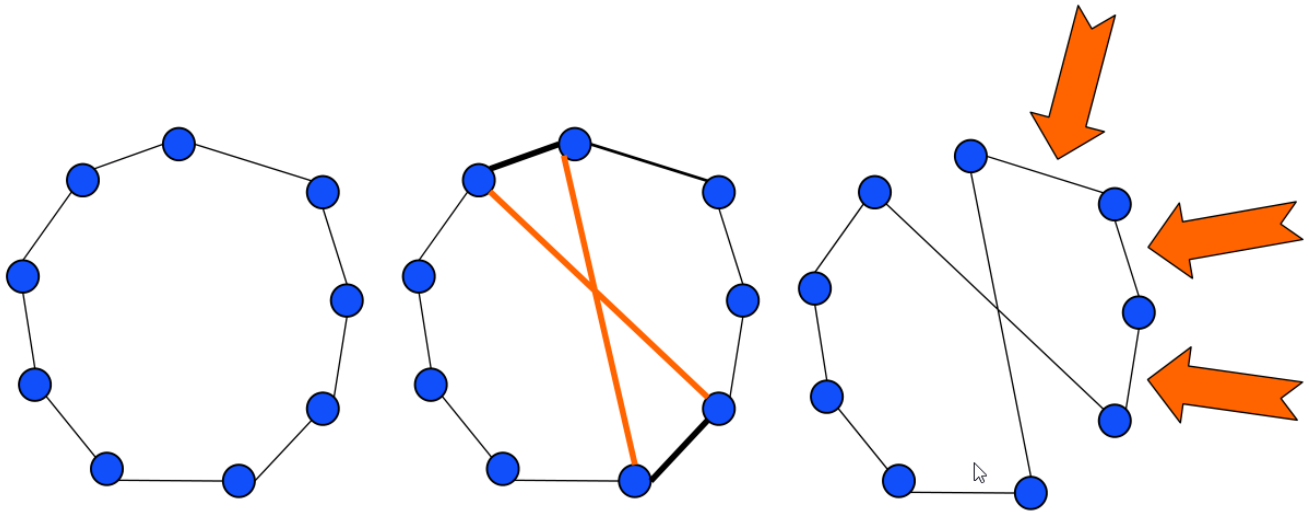
- Notation: $[E_k, L_k]$ time window on start of delivery at k
- Auxiliary information: e_k earliest time delivery can take place at k; l_k latest time delivery can take place at k
- Feasible()
  - $d_j < Q - q_r$
  - $e_j = \max\{E_j, e_{i-1} + t_{i-1,j}\} \& l_j = \min\{L_j, l_i - t_{j,i}\}$
  - $e_j < l_j$
- Update()
  - for $k = i - 1$ to 0: $l_k = \min\{l_k, l_{k+1} - t_{k,k+1}\}$
  - for $k = i$ to n: $e_k = \max\{e_k, e_{k-1} - t_{k-1,k}\}$

路径长度的可行性

- Notation: T planning horizon, L route duration limit
- Auxiliary information: e_0 earliest start time route $e_0 = \max\{0, e_{n+1} - L\}$; l_{n+1} latest completion time route $l_{n+1} = \min\{l_0 + L, T\}$; f_i total travel time from i until the end ; b_i total travel time from the beginning until i
- Feasible()
  - $e_{n+1} = \max\{e_{n+1}, e_j + t_{j,i} + f_i\}$, $e_0 = \max\{e_0, e_{n+1} - L\}$
  - $l_0 = \min\{l_0, l_j - t_{i-1,j} - b_{i-1}\}$, $l_{n+1} = \min\{l_{n+1}, l_0 + L\}$
  - $d_j < Q - q_r$
  - $e_j < l_j$
  - $e_0 < l_0$ & $l_{n+1} > e_{n+1}$
- Update()
  - for $k = i - 1$ to 0: $l_k = \min\{l_k, l_{k+1} - t_{k,k+1}\}$
  - for $k = i$ to n: $e_k = \max\{e_k, e_{k-1} - t_{k-1,k}\}$
  - if e_0 updated: for $k = 0$ to $n - 1$: $e_k = \max\{e_k, e_{k-1} - t_{k-1,k}\}$
  - if l_{n+1} updated: for $k = n + 1$ to 0: $l_k = \min\{l_k, l_{k+1} - t_{k,k+1}\}$

本节目录

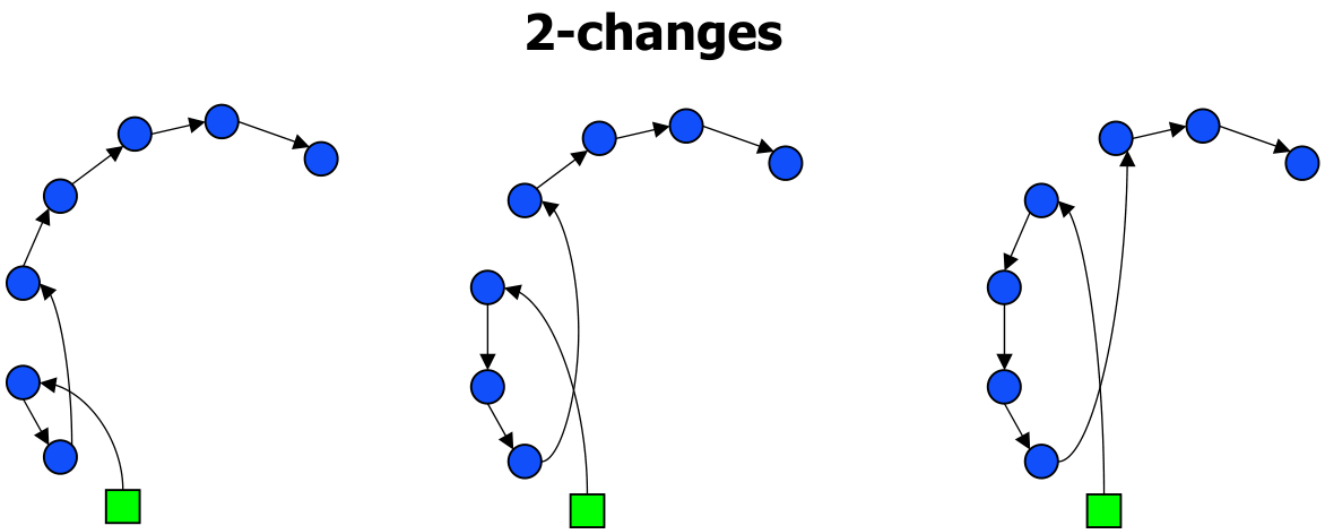
2-change 回顾



Figure

任取两点的话，部分子路径的顺序可能会变成逆向

Lexicographic search (字典式搜索)



Figure

全局变量

Let $U = \{u_1, u_2, \ldots, u_k\}, \quad V = \{v_1, v_2, \ldots, v_l\}$

- Total travel time $T(U) = \sum_{i=1}^{k-1} t(u_i, u_{i+1})$
- Earliest delivery time of the last node $E(u_k)$ assuming u_1 is left at the opening of its time window $\max_{u_i \in U}\{E_i + T(u_i, \ldots, u_k)\}$
- Latest delivery time $L(u_1)$ such that the path remains feasible $\min_{u_i \in U}\{L_i - T(u_1, \ldots, u_i)\}$

路径连接

- $T(U \bigcup V) = T(U) + t(u_k, v_1) + T(V)$
- $E(v_l) = \max\{E(u_k) + t(u_k, v_1) + T(V), E(v_l)\}$
- $L(u_1) = \min\{L(u_1), L(v_1) - T(V) - t(u_k, v_1)\}$

路径连接 (cont.)

- Earliest delivery at v_1 : $E(u_k) + t(u_k, v_1)$
- Earliest delivery at v_l : $E(u_k) + t(u_k, v_1) + T(V)$

路径连接 (cont.)

- Latest delivery at u_k : $L(v_1) - t(u_k, v_1)$
- Latest delivery at u_1 : $L(v_1) - t(u_k, v_1) - T(U)$

一些观察

- The set of global variables makes it possible to test feasibility of an exchange in constant time
- The lexicographic search strategy makes it possible to maintain the correct values for the set of global variables in constant time

本节目录

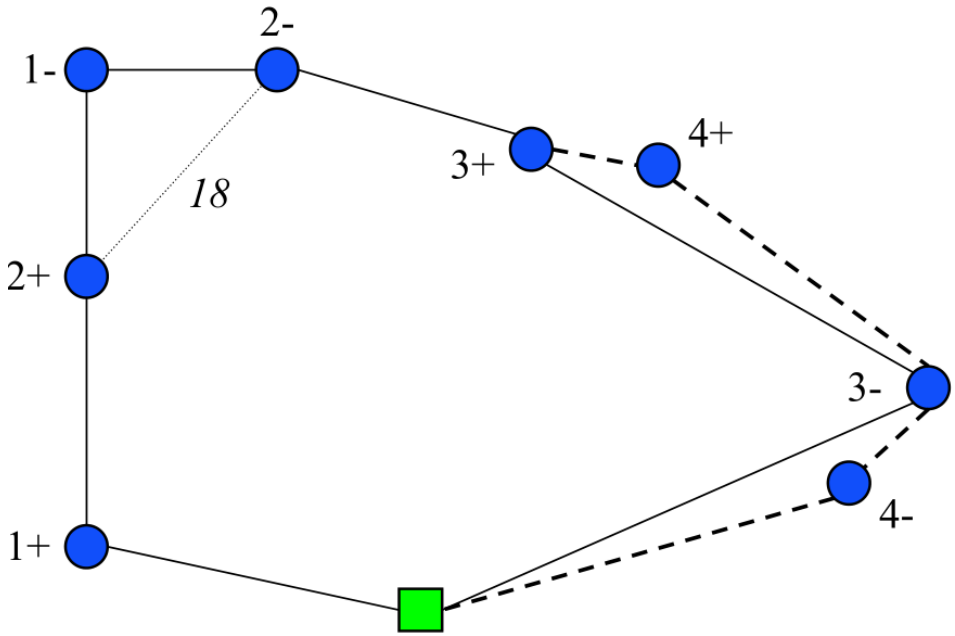
Dial-a-Ride Problem

- Dispatch vehicles to pickup person/package at one location (origin) and deliver the person/package at another location (destination)
- Service related constraints
  - pickup window / delivery window
  - maximum wait time: Limit waiting time at a stop before departing
  - maximum ride time: Limit time between pickup and delivery

Feasibility testing

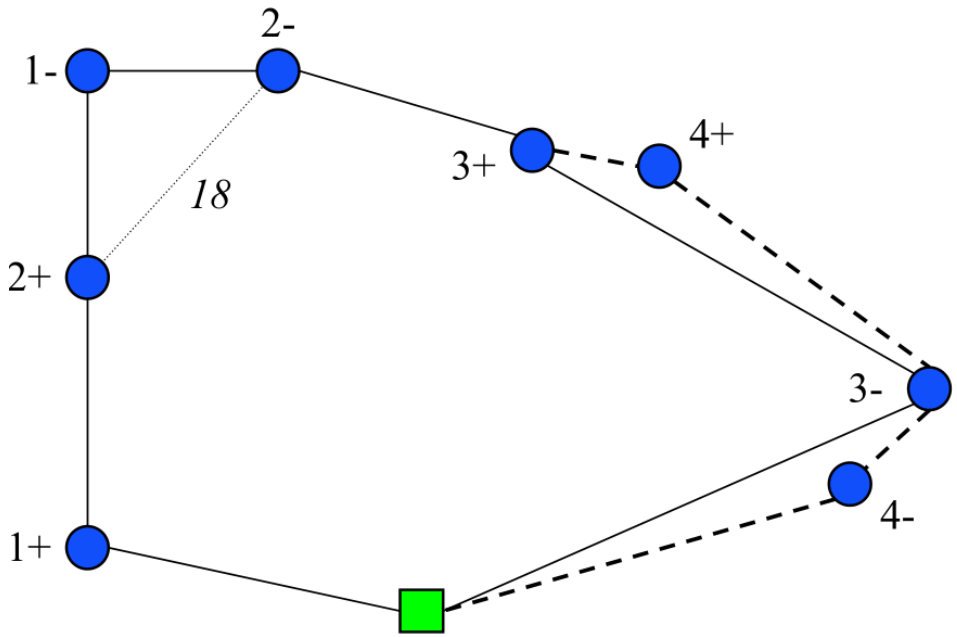
Given a sequence of pickups and deliveries does there exist a feasible schedule satisfying pickup and delivery windows, maximum wait time, and maximum ride time constraints?

Example



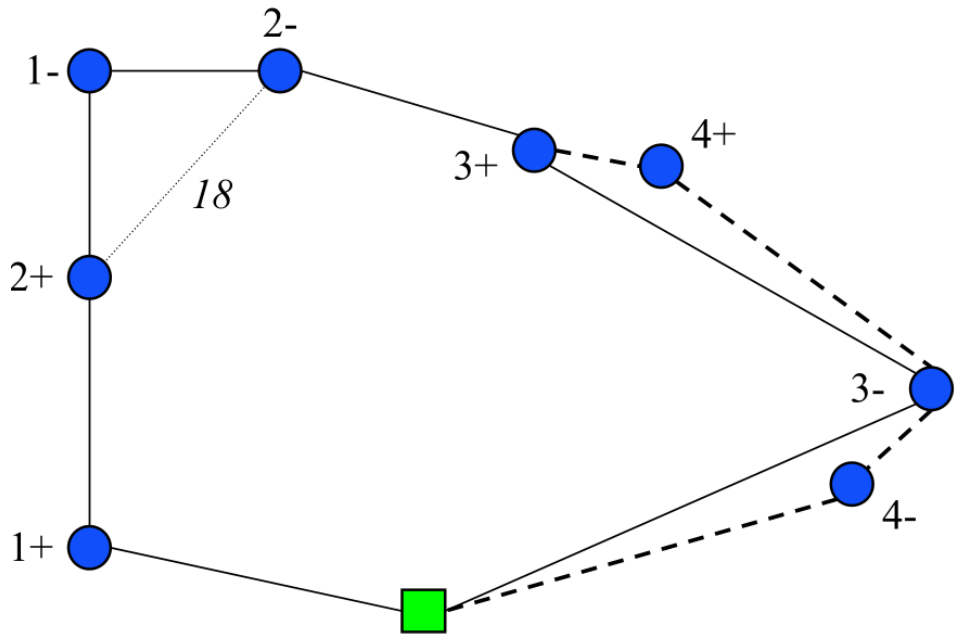
- Waiting time limit: 10
- Ride time limit: 1.5 $\times$ ride time

Example



Stop	Early	Late	Arrival	Departure	Waiting	Travel
1+	10.15	10.3	10.15	10.15	0	20
2+	10.45	11	10.35	10.45	10	15
1-	11	11.15	11	11	0	10
2-	11.2	11.4	11.1	11.2	10	15
3+	11.4	12	11.35	11.4	5	50
3-	12.3	13	12.3	12.3	0	

Example (cont.)



Stop	Early	Late	Arrival	Departure	Waiting	Travel
1+	10.15	10.3	10.15	10.15	0	20
2+	10.45	11	10.35	10.45	10	15
1-	11	11.15	11	11	0	10
2-	11.2	11.4	11.1	11.2	10	15
3+	11.4	12	11.35	11.4	5	50
4+	12.1	12.3	11.5	12.1	**20**	40
3-	12.3	13	12.5	12.5	0	20
4-	13	13.45	13.1	13.1	0	

Example (cont.)

Stop	Early	Late	Arrival	Departure	Waiting	Travel
1+	10.15	10.3	10.15	10.15	0	20
2+	10.45	11	10.35	10.45	10	15
1-	11	11.15	11	11	0	10
2-	11.2	11.4	11.1	11.2	10	15
3+	11.4	12	11.35	11.4	5	50
4+	12.1	12.3	11.5	12.1	**20**	40
3-	12.3	13	12.5	12.5	0	20
4-	13	13.45	13.1	13.1	0	

Stop	Early	Late	Arrival	Departure	Waiting	Travel
1+	10.15	10.3	10.2	10.2	0	20
2+	10.45	11	10.4	10.5	10	15
1-	11	11.15	11.05	11.05	0	10
2-	11.2	11.4	11.15	11.25	10	15
3+	11.4	12	11.4	11.5	10	10
4+	12.1	12.3	12	12.1	10	40
3-	12.3	13	12.5	12.5	0	20
4-	13	13.45	13.1	13.1	0	

Feasibility testing

- Can be done in linear time ...
- Invariant property: No feasible schedule can have an arrival or departure time earlier than the computed arrival and departure times
- Notation:

$[e_j, l_j]$	time window
ω	waiting time limit
$\alpha t_{i_+,i_-}$	ride time limit
A_j	arrival time
D_j	departure time
L_j	latest feasible departure time

Phase I-Forward

- Account for pickup and delivery windows and maximum waiting time constraints
- Normal updates:
  $A_j = D_{j-1} + t_{j-1,j}$; $\quad D_j = \max\{e_j, A_j\}$; $\quad L_j = \min\{l_j, L_{j-1} + t_{j-1,j} + \omega\}$
- Infeasibility: $A_j > l_j$; $\quad L_j < e_j$
- Special update when $A_j + \omega < e_j$: $A_j = e_j - \omega$; $D_j = e_j$

Phase II-Backward

- Update arrival and departure times and "check" ride time constraints: Waiting time from j until the end of route W
- Normal updates:
  $D_j = A_{j+1} - t_{j-1,j}$; $\quad A_j = \max\{A_j, D_j - \omega\}$; $\quad W = W + (D_j - A_j)$
- Infeasibility: $\Delta = (D_{i_-} - D_{i_+}) - \alpha t_{i_+,i_-}$; $D_{i_+} + \Delta > L_{i_+}$; $\Delta > W$
- Special update when $\Delta > W$:
  $D_j = D_j + \Delta$; $A_j = \max\{A_j, D_j - \omega\}$; $W = W - \Delta$

Phase III-Forward

- Finalize arrival and departure times and check ride time constraints
- Normal updates: $A_j = D_{j-1} + t_{j-1,j}$; $D_j = \max\{A_j, D_j\}$
- Infeasibility (drop-off point $j = i_-$):
  $\Delta = (D_{i_-} - D_{i_+}) - \alpha t_{i_+,i_-}$; $D_{i_-} > L_{i_-}$; $\Delta > 0$

人与货物的区别辨析

- People transportation
  - delivery window $[0, l_j]$
  - no pickup window
  - waiting time at pickup only
  - different ride time limits
  - consecutive stops at same location (waiting time per location rather than stop)
- Package transportation: no waiting time limits

本节目录

本节目录

Greedy Randomized Adaptive Search Procedure (GRASP)

- Construction + Improvement
- Greedily create feasible set of routes; Improve feasible set of routes

改进：多起点邻域搜索

- independent neighborhood searches
- starting solutions obtained from a previous local optimum by a suitable perturbation method

Algorithm 2: GRASP-1

```
1 Initialization: best = ∞
2 while stop criterion not reached do
3 |   Randomly create feasible set of routes
4 |   Improve feasible set of routes (local search)
5 |   if Better than best then
6 |   |   Update best
7 |   end
8 end
```

改进：多起点邻域搜索

- independent neighborhood searches
- random starting solutions

Algorithm 3: Grasp-2

```
1 Initialization: best = ∞
2 Greedily create feasible set of routes
3 while stop criterion not reached do
4 |   Perturb feasible set of routes
5 |   Improve feasible set of routes (local search)
6 |   if Better than best then
7 |   |   Update best
8 |   end
9 end
```

Greedy algorithm

- Constructs a solution one element at a time:
  - Define candidate elements
  - Apply greedy function to each candidate element
  - Rank candidate elements according to greedy function value
  - Add best ranked element to solution

Semi-greedy algorithm

- Constructs a solution one element at a time:
  - Define candidate elements
  - Apply greedy function to each candidate element
  - Rank candidate elements according to greedy function value
  - Place well-ranked elements in a restricted candidate list (RCL)
  - Select an element from the RCL at random and add it to the solution

Restricted Candidate List

- Cardinality based:
  - Place k best candidates in RCL
- Value based I:
  - Place all candidates having greedy value better than $\alpha\times$ max value in RCL (with $0 \leqslant \alpha \leqslant 1$)
- Value based II:
  - Place all candidates having greedy value better than min value $+ \alpha\times$ (max value - min value) in RCL (with $0 \leqslant \alpha \leqslant 1$)

Semi-greedy

Algorithm 4: Grasp-2

```
1 Initialization: best = ∞
2 while stop criterion not reached do
3 |   Semi-greedily create feasible set of routes
4 |   if Better than best then
5 |   |   Update best
6 |   end
7 end
```

Greedy Randomized Adaptive Search Procedure

Algorithm 5: Grasp-2

```
1 Initialization: best = ∞
2 while stop criterion not reached do
3 |   Semi-greedily create feasible set of routes
4 |   Improve feasible set of routes
5 |   if Better than best then
6 |   |   Update best
7 |   end
8 end
```

GRASP

- GRASP tries to capture good features of greedy & random constructions
- Iteratively
  - samples solution space using a greedy probabilistic bias to construct a feasible solution
  - applies local search to attempt to improve upon the constructed solution

本节目录

Neighborhood search - observations

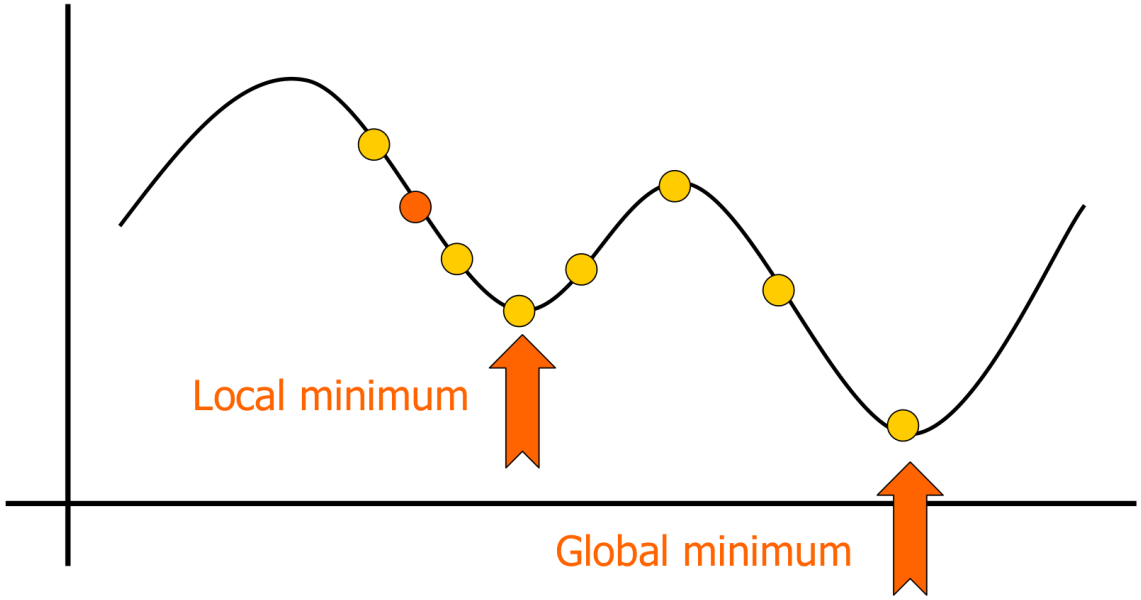
- Weakness:
  - looks only one step ahead, and may get trapped in a bad local optimum
- Strength:
  - Fast and easy to implement

Neighborhood search - sophisticated enhancements

- Goal: much:
  - Increase quality of solution
  - Do not increase time to find solution too
- Tabu Search
- Large Scale Neighborhood Search

本节目录

Tabu search



Figure

- Strategy to escape from a local optimum and continue the search
- Implementation
  - Best move is always performed
  - Avoid cycling using short-term memory
  - Attributes of recent solutions stored in tabu list
  - Moves involving attributes in tabu list are discarded (tabu)
-

Short-term Memory

- Tabu list
  - Tabu list size - maximum number of attributes stored in the list (FIFO)
  - Tabu list tenure - maximum number of iterations attribute remains in the list
- Tabu list
  - Last t moves
- Frequency-based
  - Number of times a specific move is performed
  - Penalize moves with higher frequency

Intensification and Diversification

- Intensification
  - Intensify the search in promising regions
- Diversification
  - Diversify the search across contrasting regions
- Examples
  - Varying the tabu list size
  - Adjusting the cost structure

Observations

- Tabu search can be highly effective
- Tabu search can be prohibitively time consuming → Remedy: speed up neighborhood search

Granular Tabu Search

- Reduce the number of moves evaluated at each iteration
- Routing and scheduling problems:
  - Long connections are unlikely to be part of an optimal solution

Granular neighborhoods

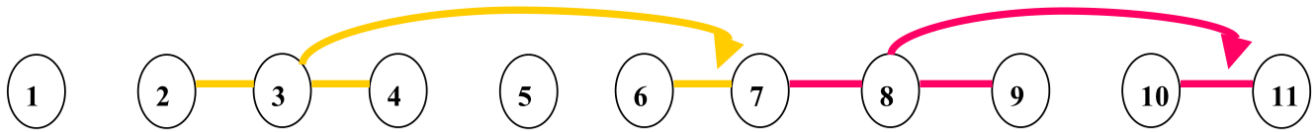
- Restriction of ordinary neighborhoods
  - Consider only connections whose cost is below a threshold
  - Consider only moves involving promising connections
  - Threshold: $v \times (UB/n)$, v sparsification parameter; UB/n average cost of connection in solution
  - Intensification/diversification tool
    - small $v \rightarrow$ intensification
    - large $v \rightarrow$ diversification

Vehicle routing problem

- Set of connections:
  - connections of the current and best solution
  - connections involving the depot
  - connections with costs less than threshold
  -
- Connections used as move generators
- Savings heuristic
- 1-relocate, 2-relocate, swap, 2-change
- Tabu tenure: random in [5,10]
- Granularity based intensification/diversification
  - intensification: v in $[1, 2]$
  - diversification: no improvement in $15 \times n$ iterations, then $v = 5$ for n iterations

本节目录

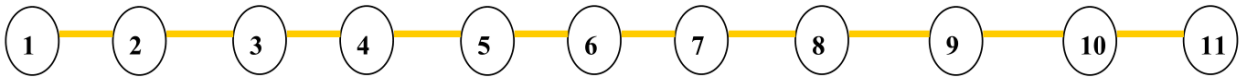
Compounded 1-relocate



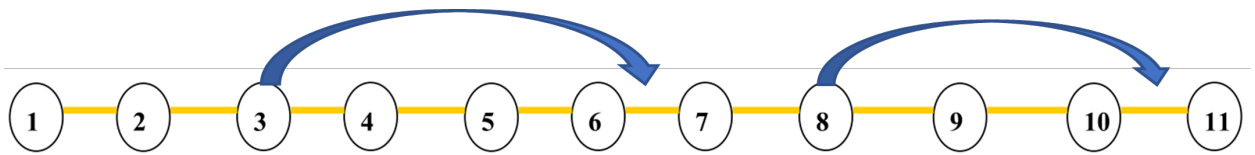
- Given the TSP tour T = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
- The new TSP tour T' = (1, 2, **4, 5, 6, 3**, 7, **9, 10, 8**, 11)
- The size of the 1-relocate neighborhood is $O(n^2)$
- The size of the compounded independent 1-relocate neighborhood is $\Theta(1.7549^n)$
  (Proof is by solving a recursion for the number of paths from 1 to n+1)

Improvement Graph

- T = (1,2,3,4,5,6,7,8,9,10,1)



- Construct improvement graph



$$c_{1,2} = 0$$

$$c_{2,7} = -(d_{2,3} + d_{3,4} + d_{6,7}) + (d_{2,4} + d_{6,3} + d_{3,7})$$

$$c_{7,11} = -(d_{7,8} + d_{8,9} + d_{10,1}) + (d_{7,9} + d_{10,8} + d_{8,1})$$

- Only forward arcs are allowed
- Node 1 is always kept fixed
- Find shortest path from 1 to n+1 in $O(n^2)$ time
- Negative cost shortest path implies an improving move

Compounded swap

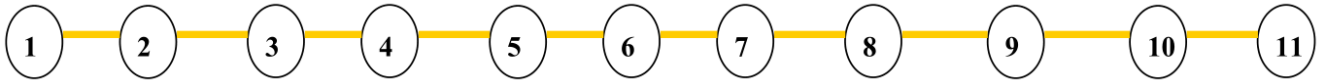


Figure: T = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)

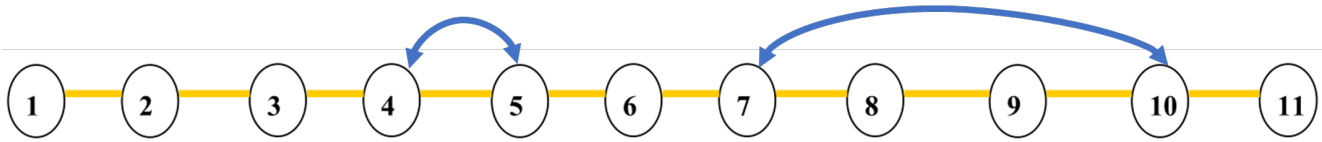


Figure: T' = (1, 2, 3, 5, 4, 6, 10, 8, 9, 7, 11)

Compounded 2-change

Figure: T = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)

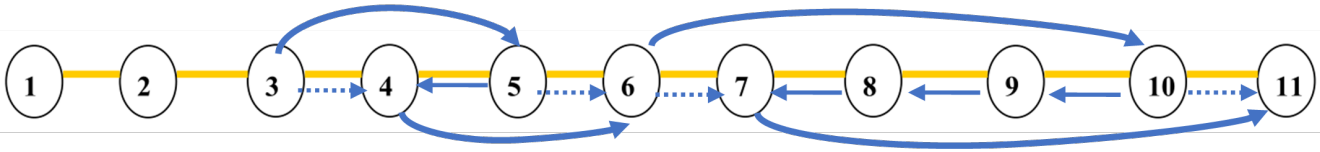


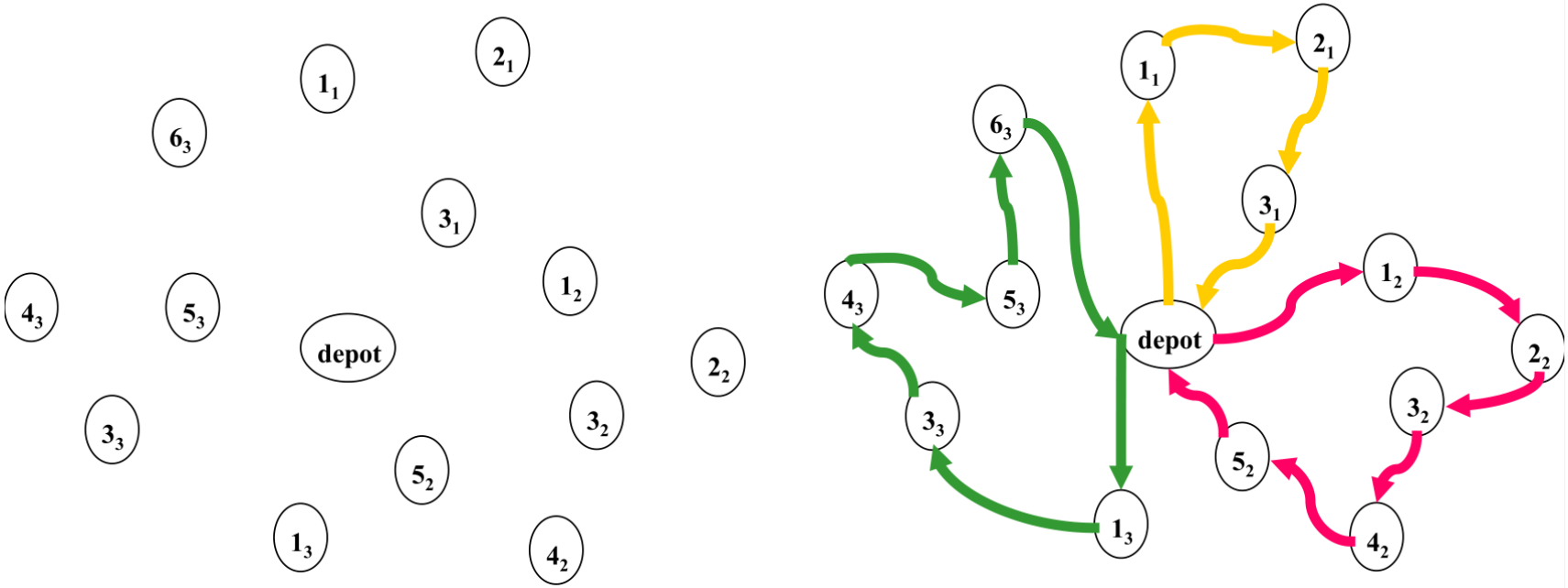
Figure: T' = (1, 2, 3, 5, 4, 6, 10, 9, 8, 7, 11)

Moreover ...

- T = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1)
- one could conduct the following moves at once
  - swap 3 and 4
  - change arc 9-10 and 12-1 to 9-12 and 10-1
  - relocate 6 to the mid of 8 and 9
- T' = (1, 2, 4, 3, 5, 7, 8, 6, 9, 12, 11, 10, 1)

本节目录

VRP 实例



表示方式–多回路表示

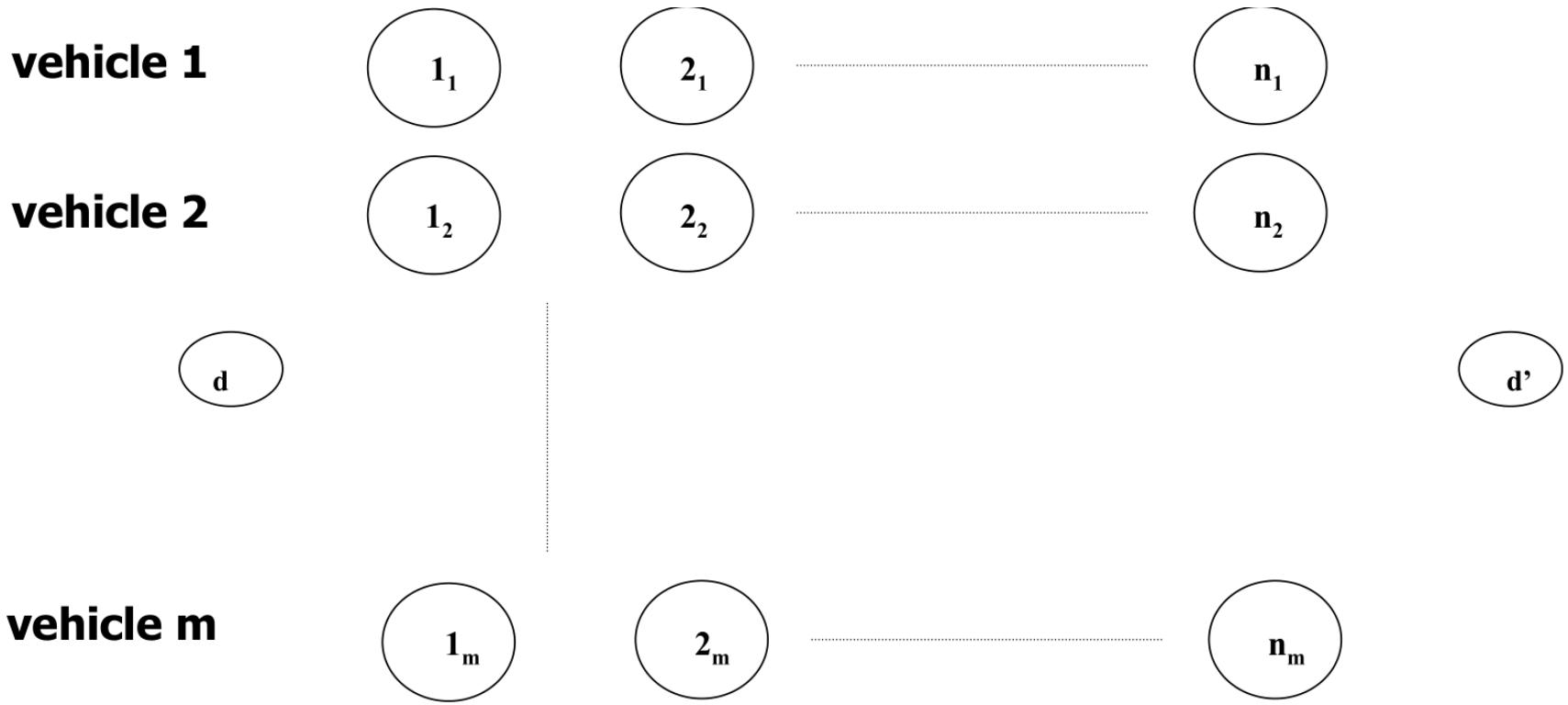


Figure: Multi tour representation

表示方式–单回路表示

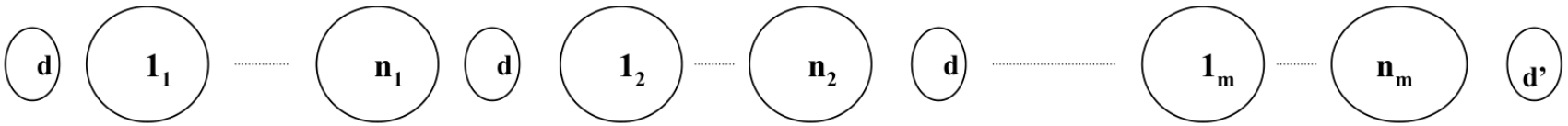
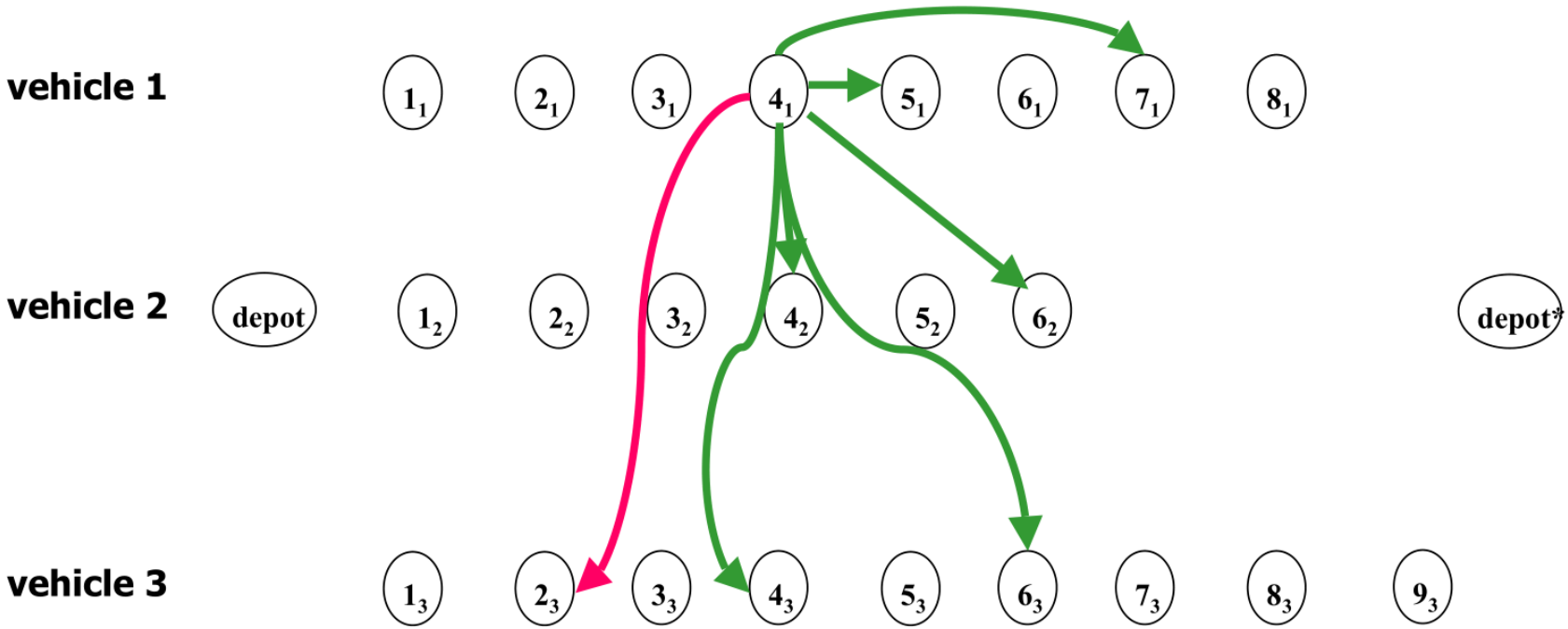


Figure: Single tour representation

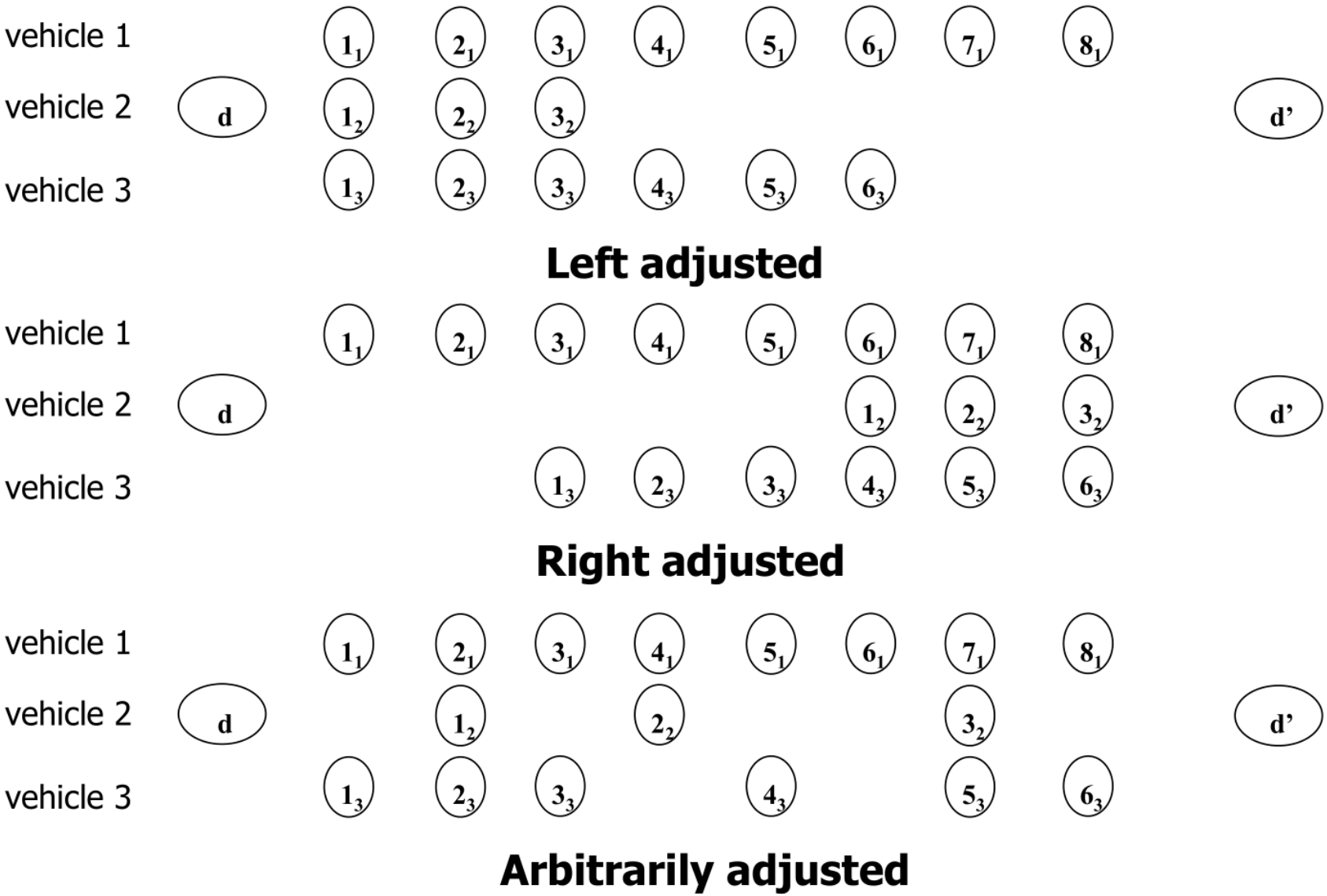
- Improvement graph is analogous to the TSP improvement graph
- For every ordering of vehicle one obtains a different neighborhood

多回路表示的问题

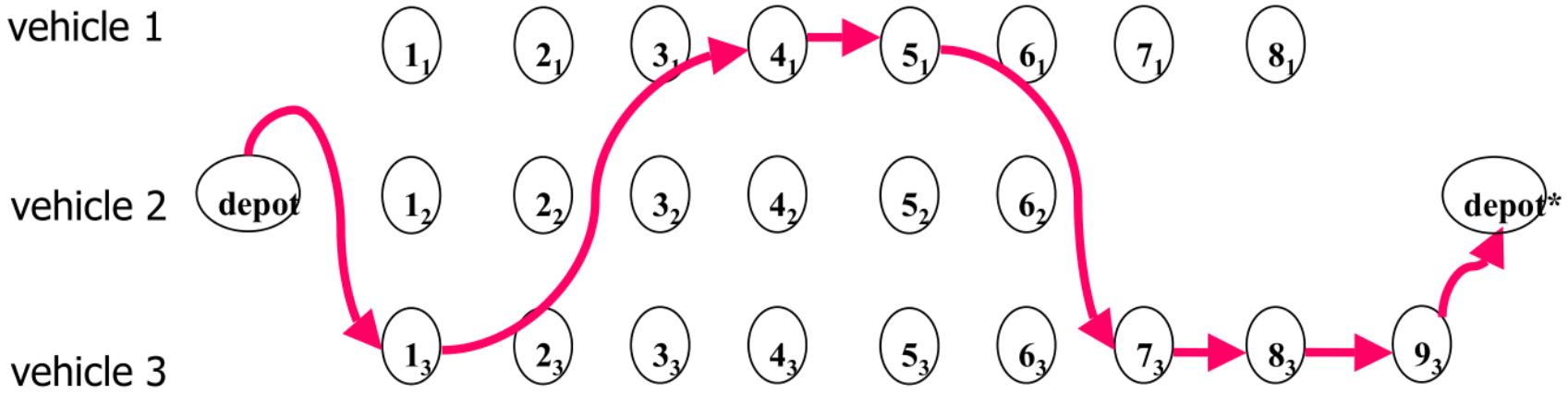


- The cost structure is not well defined for arc $(4_1, 2_3)$
- Establish an alignment scheme to define and allow only forward arcs

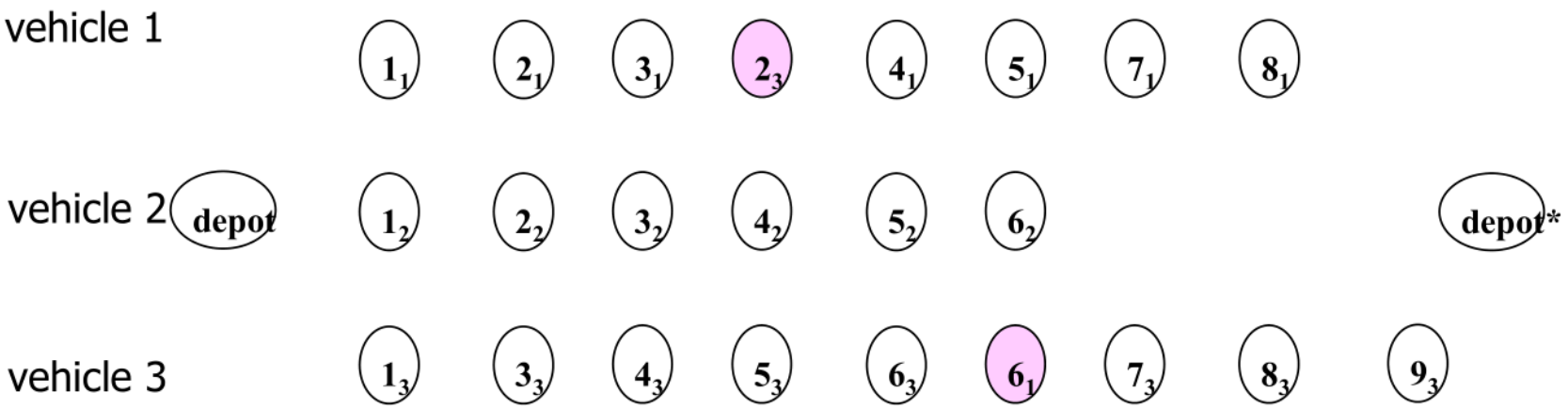
校正方法


vehicle 1
vehicle 2
vehicle 3
d
d'
Left adjusted
Right adjusted
Arbitrarily adjusted

多回路改进图



After applying the exchanges implied by the shortest path:



多回路邻域

- Additional flexibility:
  - shortest path from left to right
  - shortest path from right to left
- Additional complexity:
  - moves no longer independent due to capacity and distance restriction
- Constructing improvement graph and finding shortest path take O(n2) time

容量限制的解决方法

- For each node keep a working capacity label of each vehicle as well as a distance label
- available capacity$[k]$ = available capacity of vehicle k in current solution
- working capacity$[i, k]$ = available capacity$[k]$ + effects of changes corresponding to shortest path to i
- allow only feasible arcs with respect to working capacity in shortest path

搜索多回路和单回路图的复杂度

- Complexity of the search is $O(n^2 + nm)$
  - $O(n^2)$ for creating the improvement graph and running the shortest path algorithm
  - $O(nm)$ for updating the labels at each node once after all the incoming arcs to the node is considered

Acknowledgment

This lecture note is based on Prof. Martin Savelsbergh's tutorial on VRP. It is used for non-commercial education.