CARP Problems

Yunwen Lei

Southern University of Science and Technology

leiyw@sustc.edu.cn

October 31, 2018

Basic Procedure

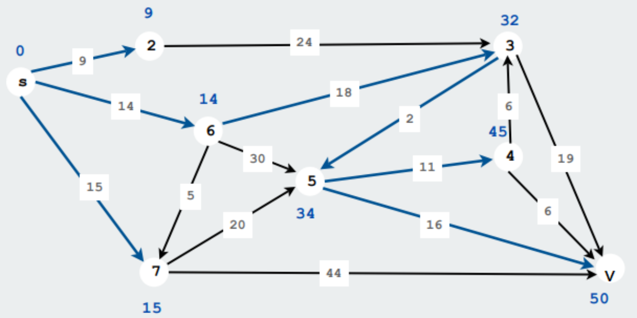
- First step: preparation
- Second step: construction
- Third step: improvement

Single-source shortest-paths

Given: weighted digraph, single source s

Distance from s to v: length of the shortest path from s to v

Goal: find distance (and shortest path) from s to **every** other vertex



Dijkstra's Algorithm

S: set of vertices for which the shortest path length from s is **known**

The **complement** $\bar{S}$ consists of nodes whose shortest path length to s is unknown

Invariant: for v in S, dist[v] is the length of the shortest path from s to v

Initialize S to s, dist[s] to 0, dist[v] to ∞ for all other v

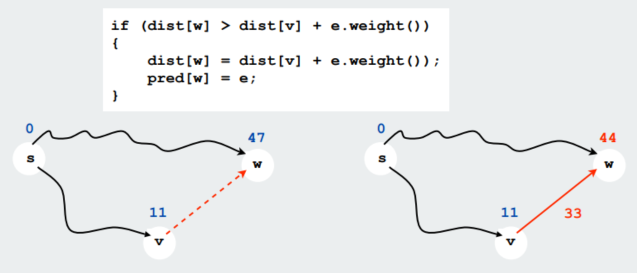
Repeat until S contains all vertices connected to s

- find v in $\bar{S}$ with minimal dist[v]
- add v to S
- relax along any edge starting from v and ending at a node in $\bar{S}$

Edge Relaxation

For all v, dist[v] is the length of some path from s to v
Relaxation along edge e from v to w

- dist[v] is the length of **shortest** path from s to v
- dist[w] is the length of **some** path from s to w
- if v-w gives a shorter path to w through v, update dist[w]



Relaxation sets dist[w] to the length of a **shorter** path from s to w (if v-w gives one)

Distance Between Any Two Points

- Apply Dijkstra's Algorithm to each node
- Flod Algorithm

Universal Path Scanning Algorithm

Copy all the arcs to the unallocated list, assuming the list is named **free**
Repeat the following steps to generate a path:

- The starting point is 1 (the specified position of the **depot**)
- Repeatedly add the task **meeting the capacity limit** and the **minimal distance** from the end point of the previous task.

  If there are tasks with the same distance, apply other preferred criteria (5 on the next page) to pick a relatively better task (or randomly choose equidistant task)
- Back to the starting point if no task can join the path under the constraint.

Algorithm 7.2 – Path-Scanning for one priority rule

```
k ← 0
copy all required arcs in a list free
repeat
    k ← k + 1; R_k ← ∅; load(k), cost(k) ← 0; i ← 1
    repeat
        d̄ ← ∞
        for each u ∈ free | load(k) + q_u ≤ Q do
            if d_{i,beg(u)} < d̄ then
                d̄ ← d_{i,beg(u)}
                ū ← u
            else if (d_{i,beg(u)} = d̄) and better(u, ū, rule)
                ū ← u
            endif
        endfor
        add ū at the end of route R_k
        remove arc ū and its opposite ū + m from free
        load(k) ← load(k) + q_ū
        cost(k) ← cost(k) + d̄ + c_ū
        i ← end(ū)
    until (free = ∅) or (d̄ = ∞)
    cost(k) ← cost(k) + d_{i1}
until free = ∅
```

Different Rules Applicable

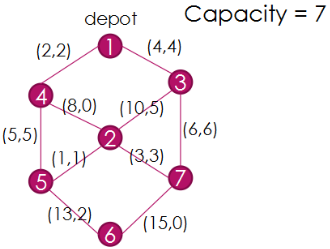
1. maximize the distance from the task to the depot;
2. minimize the distance from the task to the depot;
3. maximize the term **dem(t)/sc(t)**, where **dem(t)** and **sc(t)** are demand and serving cost of task t, respectively;
4. minimize the term **dem(t)/sc(t)**;
5. use **rule 1** if the load of the vehicle is less than half of the capacity, otherwise use **rule 2**

You can choose one rule to further select candidate tasks. You can apply **rule 1** for the first solution, apply **rule 2** for the second solution, and so on.

An Example

c[][]:

	1	2	3	4	5	6	7
1	0	8	4	2	7	20	10
2	8	0	9	6	1	14	3
3	4	9	0	6	10	21	6
4	2	6	6	0	5	18	9
5	7	1	10	5	0	13	4
6	20	14	21	18	13	0	15
7	10	3	6	9	4	15	0



Path Scanning: Initialization

free:	(1,4)	(1,3)	(4,5)	(5,6)	(2,3)	(2,5)	(2,7)	(3,7)
	(4,1)	(3,1)	(5,4)	(6,5)	(3,2)	(5,2)	(7,2)	(7,3)

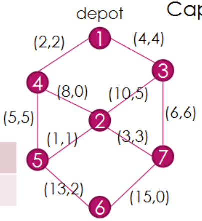
R1: ∅

load(1) = 0
cost(1) = 0
i = 1

Path Scanning: Route-1, Iteration-1

(1, 4) and (1, 3) are the 2 most recent tasks. If you select the task according to **rule 5**, you can select the task (1, 3), and (1, 3) has a far **return distance**.

Capacity = 7



free:								
	(1,4)	~~(1,3)~~	(4,5)	(5,6)	(2,3)	(2,5)	(2,7)	(3,7)
	(4,1)	~~(3,1)~~	(5,4)	(6,5)	(3,2)	(5,2)	(7,2)	(7,3)
R1 :	(1,3)							

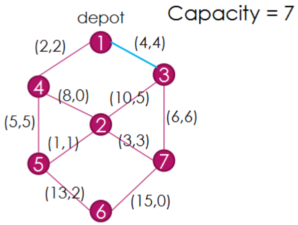
load(1) = 4
cost(1) = 4
i = 3

Path Scanning: Route-1, Iteration-2

We have capacity 3 left for which the available task is

$$(1,4)\ (4,1)\ (2,5)\ (5,2)\ (2,7)\ (7,2)\ (5,6)\ (6,5).$$

The one closest to 3 is the task $(1,4)$.



free:

~~(1,4)~~	(4,5)	(5,6)	(2,3)	(2,5)	(2,7)	(3,7)
~~(4,1)~~	(5,4)	(6,5)	(3,2)	(5,2)	(7,2)	(7,3)

R1 :

(1,3)	(1,4)

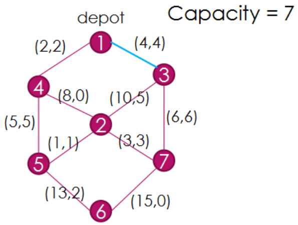
load(1) = 6
cost(1) = 4 + c[3][1] + 2 = 10
i = 4

Path Scanning: Route-1, Iteration-2

We have capacity 3 left for which the available task is

$$(1,4)\ (4,1)\ (2,5)\ (5,2)\ (2,7)\ (7,2)\ (5,6)\ (6,5).$$

The one closest to 3 is the task $(1,4)$.



free:	~~(1,4)~~	(4,5)	(5,6)	(2,3)	(2,5)	(2,7)	(3,7)
	~~(4,1)~~	(5,4)	(6,5)	(3,2)	(5,2)	(7,2)	(7,3)
R1 :	(1,3)	(1,4)					

load(1) = 6
cost(1) = 4 + c[3][1] + 2 = 10
i = 4

Path Scanning: Route-1, Iteration-3

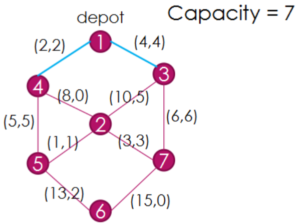
We have capacity 1 left for which the available task is

$$(2,5)\ (5,2).$$

The one closest to 4 is the task $(5,2)$.

free:	(4,5)	(5,6)	(2,3)	~~(2,5)~~	(2,7)	(3,7)
	(5,4)	(6,5)	(3,2)	~~(5,2)~~	(7,2)	(7,3)
R1 :	(1,3)	(1,4)	(5,2)			

load(1) = 7
cost(1) = 10+c[4][5]+1= 16
i = 2

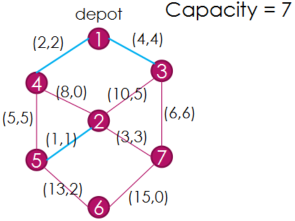


Path Scanning: Route-1, Finished

Zero capacity left.

free:	(4,5)	(5,6)	(2,3)	(2,7)	(3,7)
	(5,4)	(6,5)	(3,2)	(7,2)	(7,3)
R1 :	(1,3)	(1,4)	(5,2)		

load(1) = 7
cost(1) = 16 + c[2][1] = 16 + 8 = 24



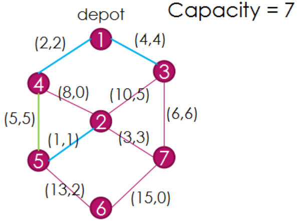
Path Scanning: Route-2, Iteration-1

The closest task to 1 is $(4, 5)$ and $c[1][4] = 2$

free:

~~(4,5)~~	(5,6)	(2,3)	(2,7)	(3,7)
~~(5,4)~~	(6,5)	(3,2)	(7,2)	(7,3)

R2 : (4,5)



load(2) = 5
cost(2) = c[1][4] + 5 = 7
i = 5

Path Scanning: Route-2, Iteration-2

We have capacity 2 left for which the available task is

$$(5,6)\ (6,5).$$

The one closest to 5 is the task $(5,6)$ and $c[5][5] = 0$

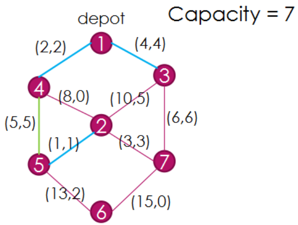
free:

~~(5,6)~~	(2,3)	(2,7)	(3,7)
~~(6,5)~~	(3,2)	(7,2)	(7,3)

R2 :

(4,5)	(5,6)

load(2) = 7
cost(2) = 7 + c[5][5] + 13 = 20

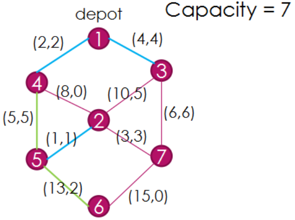


Path Scanning: Route-2, Finished

Zero capacity 2 left.

free:	(2,3)	(2,7)	(3,7)
	(3,2)	(7,2)	(7,3)
R2 :	(4,5)	(5,6)	

load(2) = 7
cost(2) = 20 + c[6][1] = 40

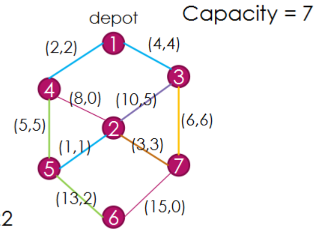


Path Scanning: Termination

Zero capacity 2 left.

free: ∅

R1:	(1,3)	(1,4)	(5,2)	load(1) = 7 cost(1) = 24
R2:	(4,5)	(5,6)		load(2) = 7 cost(2) = 40
R3:	(3,2)			load(3) = 5 cost(3) = c[1][3]+10+c[2][1]=22
R4:	(3,7)			load(4) = 6 cost(4) = c[1][3]+6+c[7][1]=20
R5:	(2,7)			load(5) = 3 cost(5) = c[1][2]+3+c[7][1]=21



Path-Scanning

- Path-scanning can guarantee a viable solution (congratulations, it will pass this game!)
- The basis of applying greedy algorithms
- Used for local search algorithm to calculate initial population

Other Construction Method

- Augment-Merge
- Ulusoys route-first cluster-secound method
- Construct-strike

Please refer to “Arc Routing” [Angel Corberan and Gilbert Laporte] P144-P149

Common Operators (Move Operator)

- Flip
- Single insertion
- Double insertion
- Swap
- 2-opt

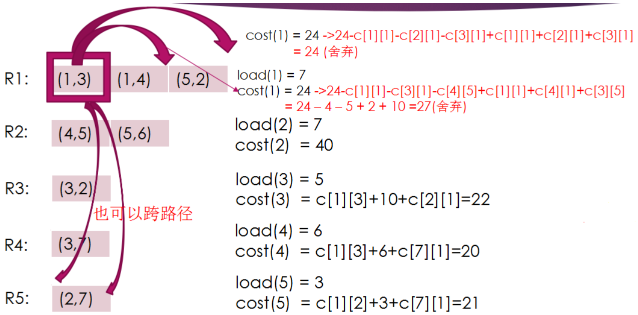
Flip

(3,1)

R1:	(1,3)	(1,4)	(5,2)	load(1) = 7 cost(1) = 24 ->24-c[1][1]-c[3][1]+c[1][3]+c[1][1]=24
R2:	(4,5)	(5,6)		load(2) = 7 cost(2) = 40
R3:	(3,2)			load(3) = 5 cost(3) = c[1][3]+10+c[2][1]=22
R4:	(3,7)			load(4) = 6 cost(4) = c[1][3]+6+c[7][1]=20
R5:	(2,7)			load(5) = 3 cost(5) = c[1][2]+3+c[7][1]=21

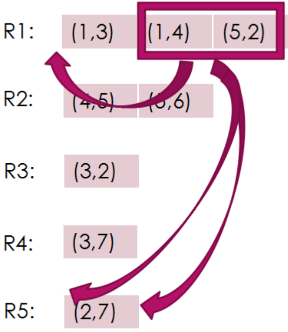
Traverse all tasks in Flip to see improvements

Single Insertion



Try to insert each individual task after other tasks in this path or after the depot, or after tasks in other route, or after depot.

Double Insertion



load(2) = 7
cost(2) = 40

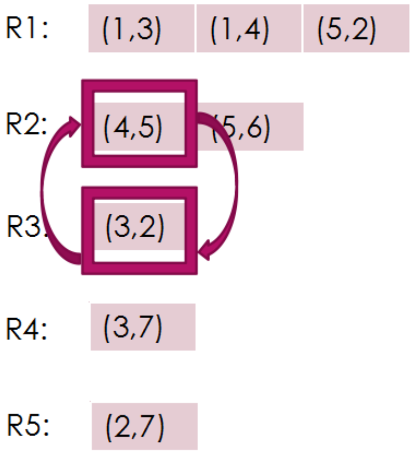
load(3) = 5
cost(3) = c[1][3]+10+c[2][1]=22

load(4) = 6
cost(4) = c[1][3]+6+c[7][1]=20

load(5) = 3
cost(5) = c[1][2]+3+c[7][1]=21

Analogous to single insertion. Try to insert two consecutive tasks after other tasks in this path or after the depot, or after tasks in other route, or after depot.

Swap



load(1) = 7
cost(1) = 24

load(2) = 7
cost(2) = 40

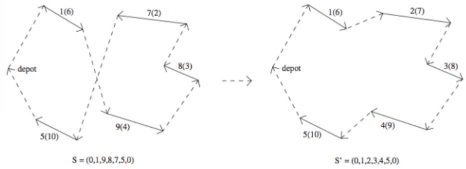
load(3) = 5
cost(3) =22

load(4) = 6
cost(4) = 20

load(5) = 3
cost(5) = 21

2-OPT

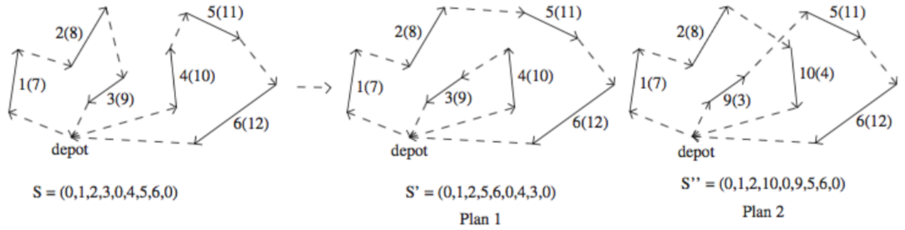
optimal for single route



4) 2-opt: There are two types of 2-opt move operators, one for a single route and the other for double routes. In the 2-opt move for a single route, a subroute (i.e., a part of the route) is selected and its direction is reversed. When applying the 2-opt move to double routes, each route is first cut into two subroutes, and new solutions are generated by reconnecting the four subroutes. Figs. 3 and 4 illustrate the two 2-opt move operators, respectively. In Fig. 3, given a solution $S = (0, 1, 9, 8, 7, 5, 0)$, the subroute from task 9 to 7 is selected and its direction is reversed. In Fig. 4, given a solution $S = (0, 1, 2, 3, 0, 4, 5, 6, 0)$, the first route is cut between tasks 2 and 3, and the second route is cut between tasks 4 and 5. A new solution can be obtained either by connecting task 2 with task 5, and task 4 with task 3, or by linking task 2 to the inversion of task 4, and task 5 with inversion of task 3. In practice, one may choose the one with the smaller cost. Unlike the previous three operators, the 2-opt operator is only applicable to edge tasks. Although it can be easily modified to cope with arc tasks, such work remains absent in the literature.

2-OPT

Optimal for double route



Problems

How to design efficient optimal operators?

- Small steps lead to trap **local optimum**?
- Large steps can miss **global optimum** when approaching it.
- No universal operator working well in all situations
- Suggestion: try small steps for **local optimum** and then escape with large steps. If get out successfully, try small steps again for **global optimum**.