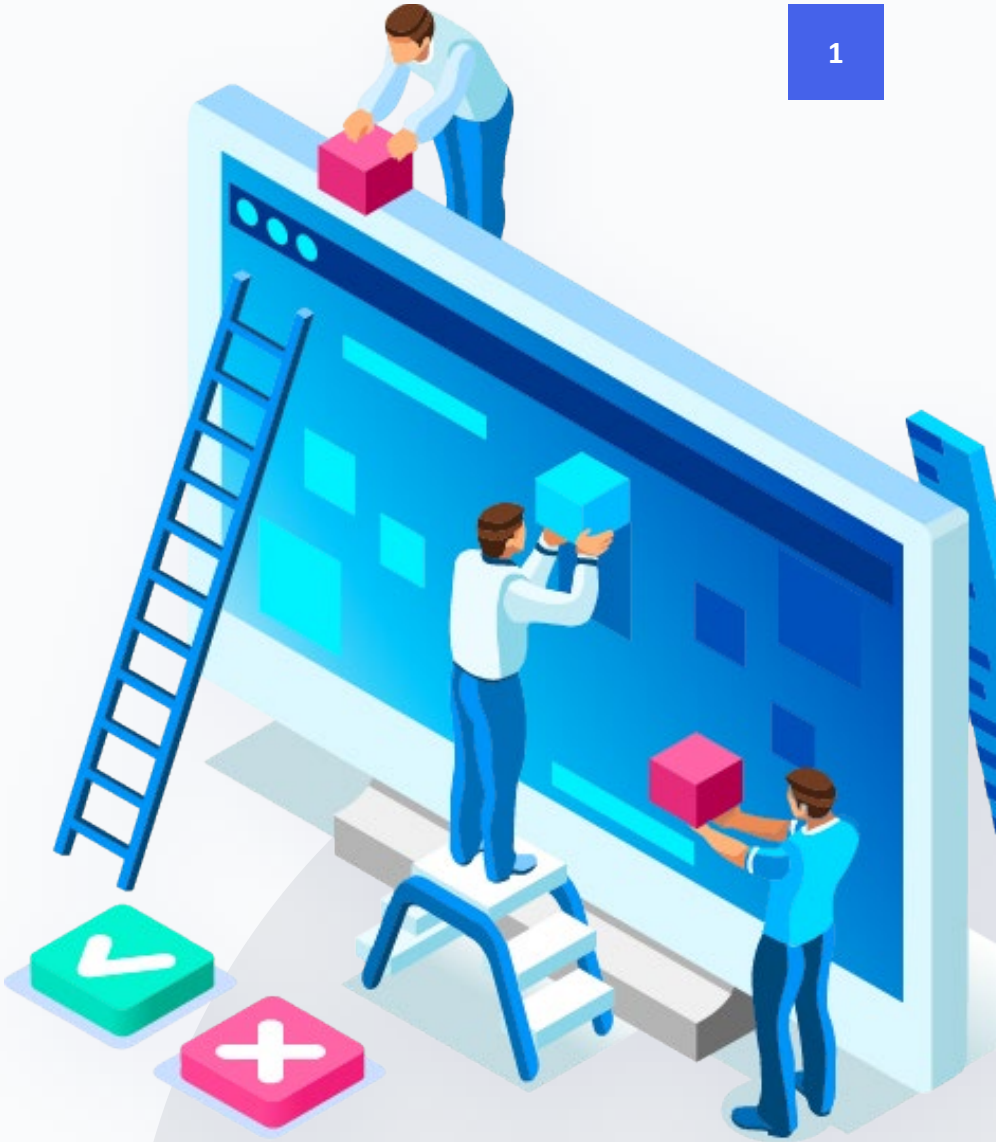
IE5600 Applied Programming | Lecture 10

Meta-heuristics I

Andrew Lim



Course Logistics – Schedule update

10:23/10

Meta-heuristics I

Simulated Annealing, Tabu Search, Variable Neighborhood Search, Large Neighborhood Search, Iterated Local Search

Homework 8 (Bonus): Constructive Algorithm for TSP – Important to practice

Group Project 2: Time Dependent VRP using the above methods

Deadline 06/11 2359hrs

Programming Test 5 – 23/10 from 1945-2100hrs On DP, Graph, Backtracking

11:30/10

Meta-heuristics II

GRASP, Multi-start methods, NGA, Memetic Algorithms, ACO, Swarm Intelligence Coding + Others

Group Project 3 – Time Dependent VRP using each of the above methods

Deadline 06/11 2359hrs

Course Logistics – Schedule update

12:06/11

Class Presentation of Project 2 and Project 3

Group Project 4 – Comparing the previous methods that have been implemented

Deadline 13/11 2359hrs

Group Project 5 – Project Summary Report

Deadline 20/11 2359hrs

13:13/11

Programming Test (Individual)

3-hour programming test – Covering Greedy, Dynamic Programming, Search, Graph and Optimization

Group to TA Assignment

	TAN RAYEN	e0175998@u.nus.edu						
	TAN SHUWEN	e0384826@u.nus.edu		CHANG LIANG	e0014813@u.nus.edu		LEOW WEE SHENT JORD	e0176071@u.nus.edu
A	EOW CHENG ZHENG	e0344189@u.nus.edu		NI XUE	e0006939@u.nus.edu		GU RUIXUE	e0251049@u.nus.edu
	YASMIN BINTE AHMAD	e0384222@u.nus.edu	D	THIRUGNANA SAMBANI	e0010871@u.nus.edu	G	CHONG WOONKIAT	e0452659@u.nus.edu
	LEE MING HOR	e0452894@u.nus.edu		NGOH SISUI	e0452722@u.nus.edu		KHOO EE QING	e0384226@u.nus.edu
	XIE XINGCHEN	e0384332@u.nus.edu		PENG JINGMING	e0383707@u.nus.edu		IRNA BINTI JUMAHAT	e0343996@u.nus.edu
	LIM WEI YANG DYLAN	e0176039@u.nus.edu		LIM ZHEN WEI ZAN	e0176128@u.nus.edu		WU TONG	e0546064@u.nus.edu
	SEAH CHOON KONG	e0030914@u.nus.edu		HU XINPING	e0344056@u.nus.edu		WANG HAI	e0008442@u.nus.edu
B	WU ZHUOCHUN	e0546183@u.nus.edu	E	HUANG XUAN	e0384269@u.nus.edu	H	LEI HAO	e0014898@u.nus.edu
	CHIN KAR KAY	e0385170@u.nus.edu		LV JISHAODONG	e0341657@u.nus.edu		TANG CHOR THENG	e0384417@u.nus.edu
	LOO WENG HENG	e0450255@u.nus.edu		TAMILARASAN SO TEYG	e0319471@u.nus.edu		SOH LI JING	e0452786@u.nus.edu
	LAI JUN GUO SCOTT	e0175273@u.nus.edu		HOONG AN SHENG SAM	e0175262@u.nus.edu		WANG XIN	e0408701@u.nus.edu
	JIAO YANG	e0450317@u.nus.edu		HUO TIANMING	e0007875@u.nus.edu		ZHOU LINLI	e0452923@u.nus.edu
C	WANG YONGJIE	e0516177@u.nus.edu	F	TOH HAN WEI	e0344017@u.nus.edu	I	LOW SIN FOU	e0344083@u.nus.edu
	KWOK JEFFERY	e0384210@u.nus.edu		DONG ZHENG	e0450318@u.nus.edu		SUTAVERAYA SURATAN	e0176486@u.nus.edu
	SHI HANG	e0450191@u.nus.edu		DOMINIQUE YEO ZONG	e0452886@u.nus.edu		KEN LIM JUNE KUANG	e0344074@u.nus.edu

Li Hongpeng – TDVRPTW (Project 1)

Zhao Huangjie – TDVRPPC (Project 2)

Yuan Yiliang – TDPDP (Project 3)

Che Yuxin – TDVRPPCTW (Project 4)

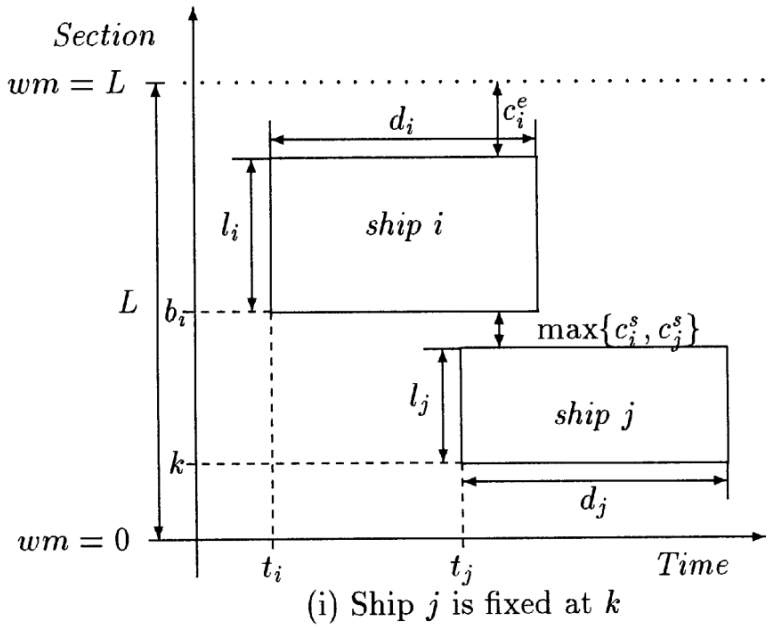
Pan Binbin – TDPDPP(Project 5)

What is a meta-heuristic?

"A metaheuristic is formally defined as *an iterative generation process* which guides a *subordinate heuristic* by combining intelligently different concepts for *exploring and exploiting* the *search space*, *learning strategies* are used to structure information in order to *find efficiently near-optimal solutions*."

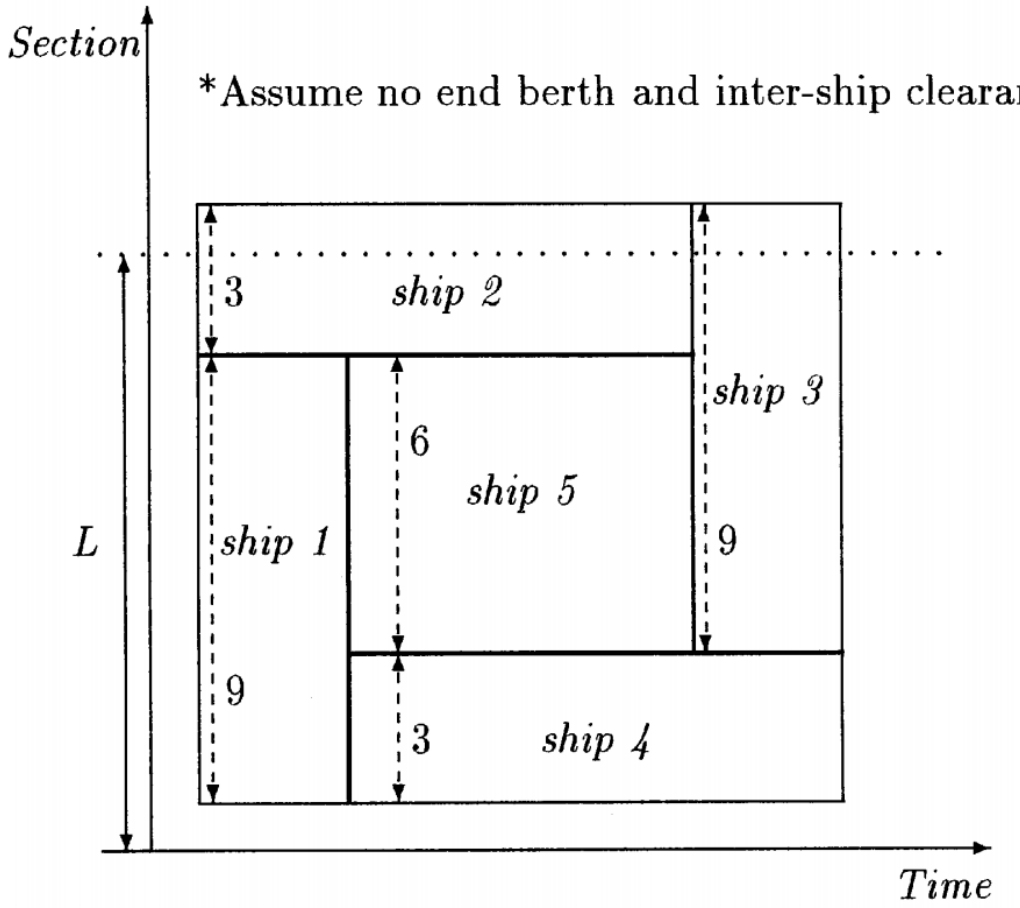
- Osman and Laporte (1996)

Revisit the Ship Berthing Problem Again

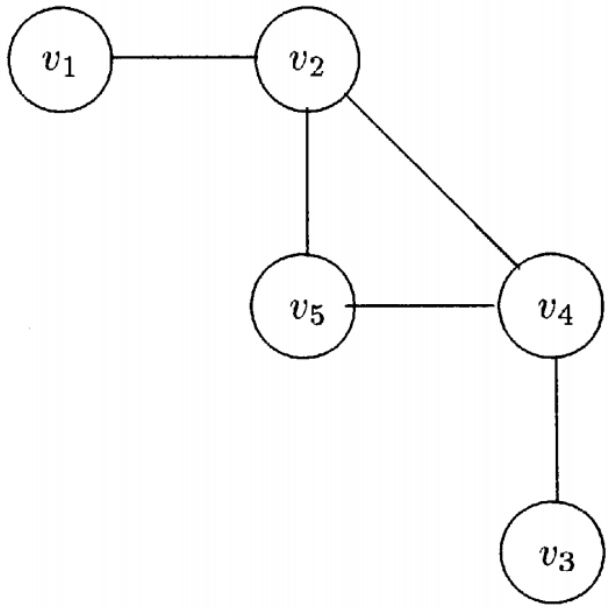


(i) Ship j is fixed at k

Problem Solving: Ship Berthing Problem – Representation

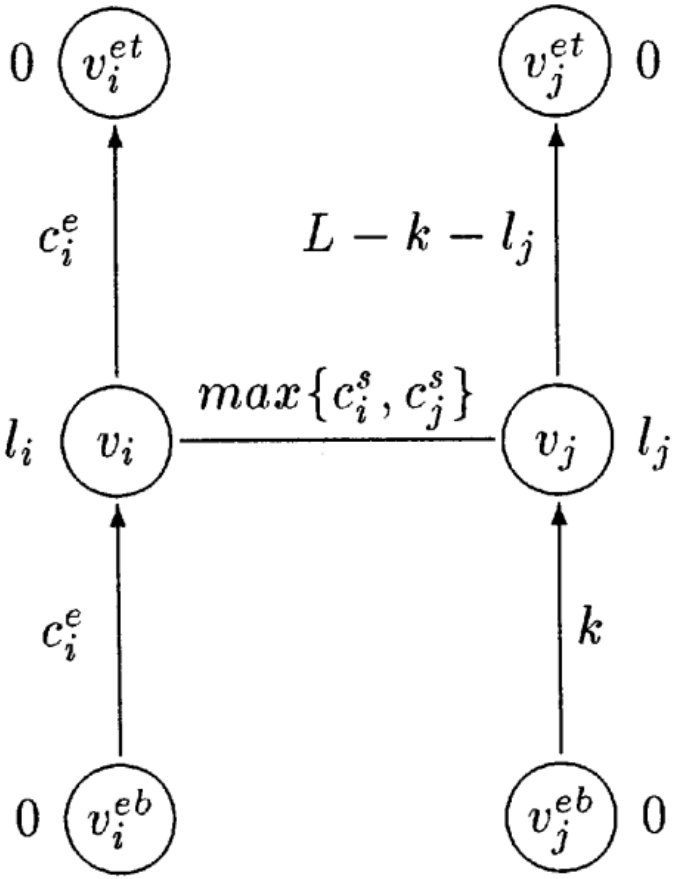


(i) Geometric Representation

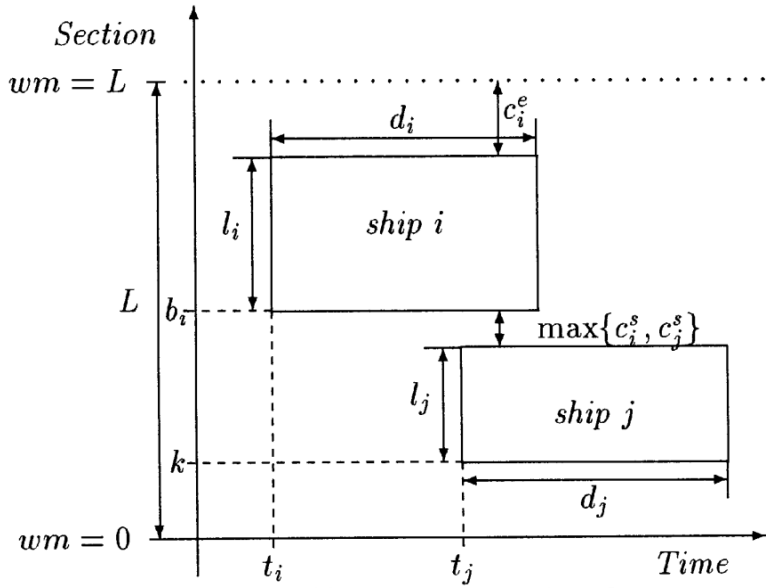


(ii) Graph Representation

Problem Solving: Handling other constraints



(ii) Graph Model



(i) Ship j is fixed at k

From Representation to an algorithm (Constructive)

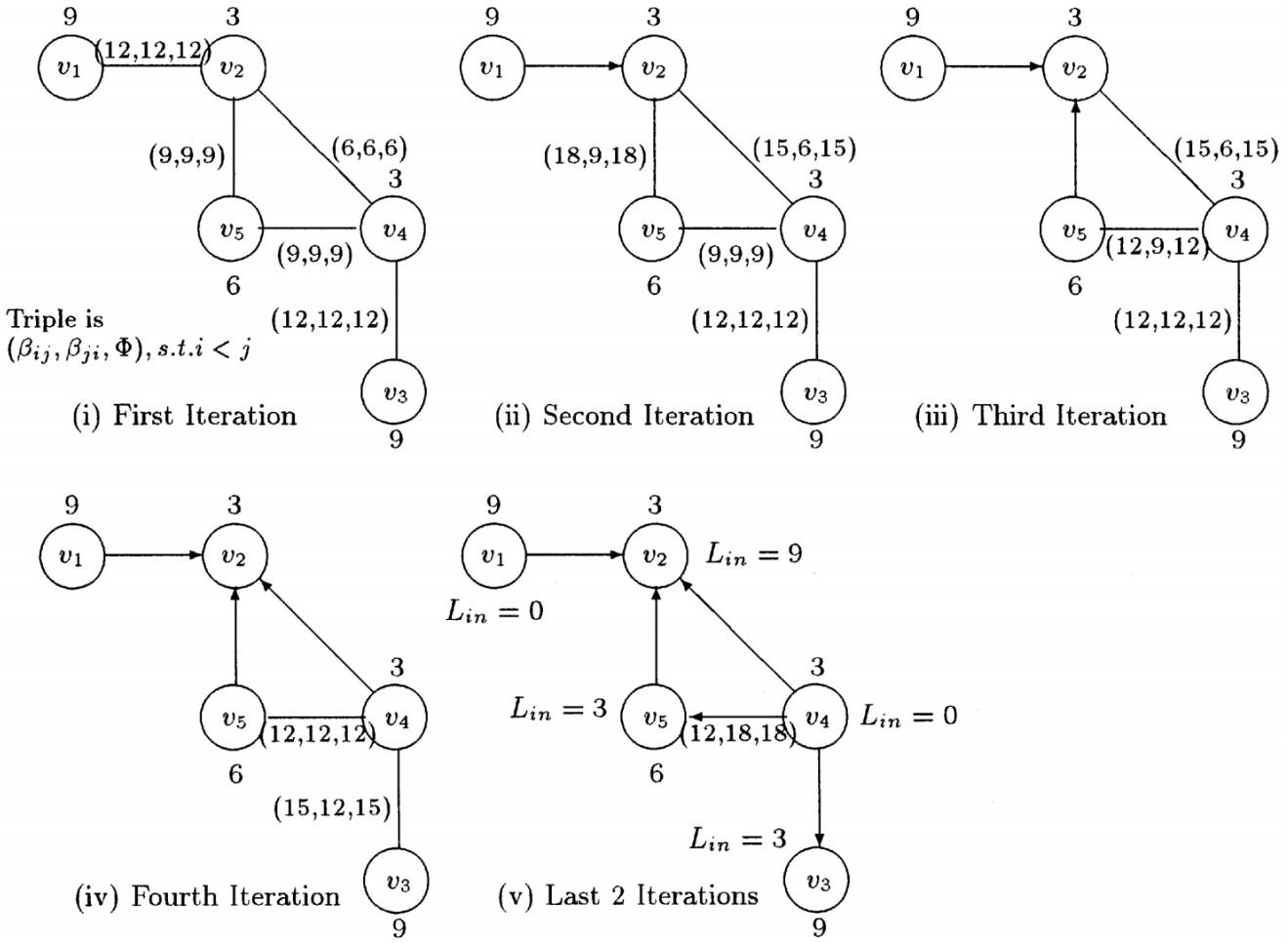
$$\beta_{ij} = \text{LongestIn}(v_i) + \text{weight}(v_i) + \text{weight}(e_{ij}) + \text{weight}(v_j) + \text{LongestOut}(v_j),$$

$$\beta_{ji} = \text{LongestIn}(v_j) + \text{weight}(v_j) + \text{weight}(e_{ij}) + \text{weight}(v_i) + \text{LongestOut}(v_i).$$

For every undirected edge, e_{ij}, the potential of edge, $\Phi(e_{ij})$ is given by $\max\{\beta_{ij}, \beta_{ji}\}$.

Algorithm BerthPlanner

Step 1: $\forall v_i \in V(\mathcal{G})$ set $LongestIn(v_i) = 0$ and $LongestOut(v_i) = 0$
Step 2: $\forall e_{ij} \in E(\mathcal{G})$ compute $\Phi(e_{ij})$
Step 3: Find the undirected edge, e_{ij}, with the highest potential $\Phi(e_{ij})$.
If there is a tie in the highest potential, pick the edge with the largest $|\beta_{ij} - \beta_{ji}|$.
Step 4: If $(\beta_{ij} < \beta_{ji})$
then set edge to go from v_i to v_j
else if $(\beta_{ij} > \beta_{ji})$
then set edge to go from v_j to v_i
else if $deg(v_i) < deg(v_j)$
then set edge to go from v_i to v_j
else set edge to go from v_j to v_i
Step 5: Update the affected longest incoming and outgoing paths of vertices
Step 6: Update the potential of affected undirected edges
Step 7: If there is an undirected edge, goto 3
Step 8: End



Constructive Heuristics

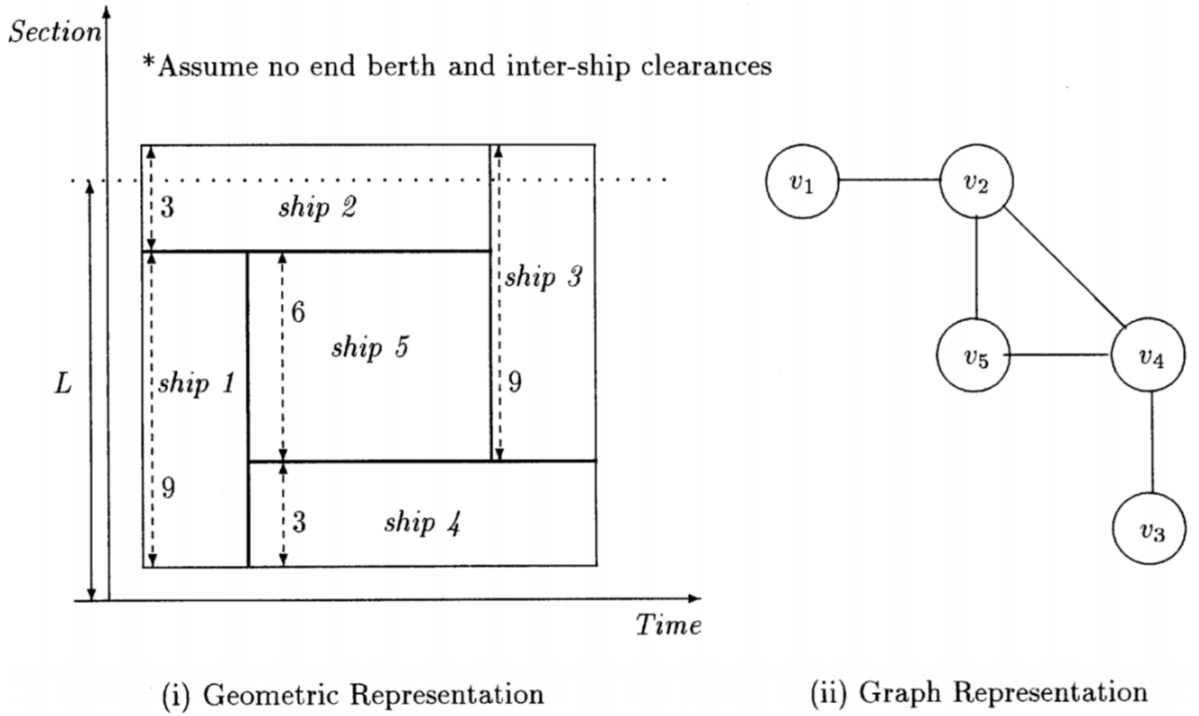
- Construct a solution as a sequence of decision choices, without revising these choices afterwards.
- Fast but seldom generate high-quality solutions.
- Can be used as a component of another iterative process – that iterative process is known as meta-heuristic

What is a meta-heuristic?

"A metaheuristic is formally defined as *an iterative generation process* which guides a *subordinate heuristic* by combining intelligently different concepts for *exploring and exploiting* the *search space*, *learning strategies* are used to structure information in order to *find efficiently near-optimal solutions*."

- Osman and Laporte (1996)

Other Representations



(i) Geometric Representation (ii) Graph Representation

- 0/1 for each edge
  - Acyclic
  - Longest incoming path is the lowest berth location
  - Needs a transformer/Decoder - DAG
- Label for each node
  - assuming every edge will go from nodes with lower labels to nodes with higher label
  - DAG transformer
- Order of packing,
  - for example given an order 4,1,5,3,2 pack the ship bottommost wherever possible
  - Scan for possible locations

DAG Transformation – Many Applications: Machine Scheduling

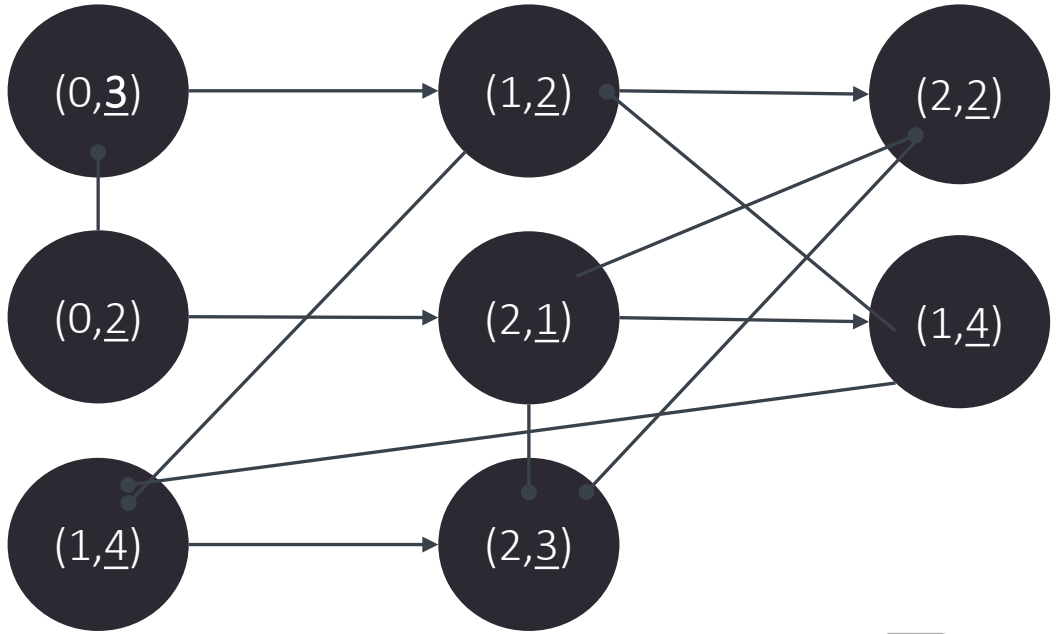
- In the job shop problem, in which each task is labeled by a pair of numbers (m, p) where m is the number of the machine the task must be processed on and p is the processing time of the task — the amount of time it requires. (The numbering of jobs and machines starts at 0.)
- job 0 = [(0, 3), (1, 2), (2, 2)]
- job 1 = [(0, 2), (2, 1), (1, 4)]
- job 2 = [(1, 4), (2, 3)]
- In the example, job 0 has three tasks. The first, (0, 3), must first be processed on machine 0 in 3 units of time. Then the second, (1, 2), must be processed on machine 1 in 2 units of time, and so on. Altogether, there are eight tasks.

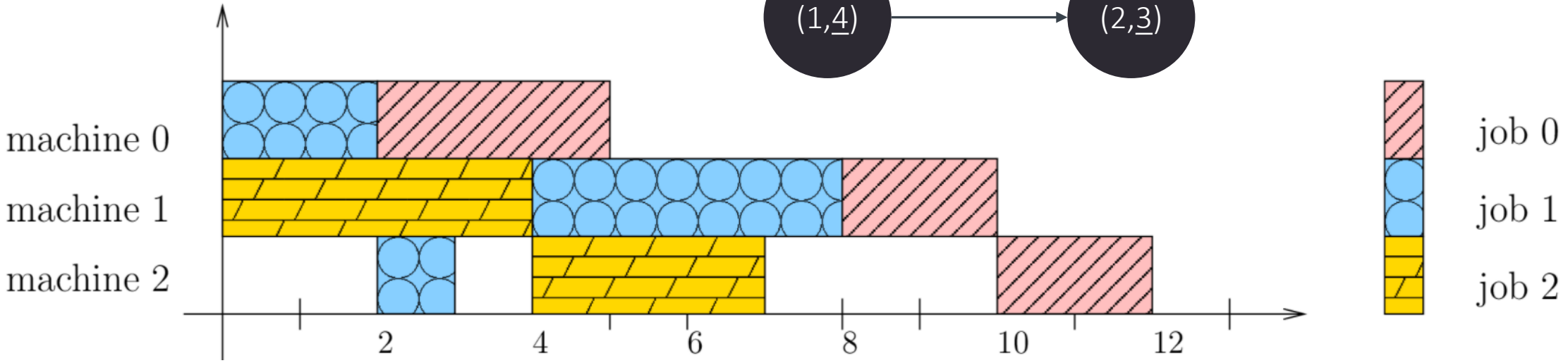
DAG Transformation – Many Applications: Machine Scheduling

job 0 = [(0, 3), (1, 2), (2, 2)]

job 1 = [(0, 2), (2, 1), (1, 4)]

job 2 = [(1, 4), (2, 3)]





Solution Representation and local search

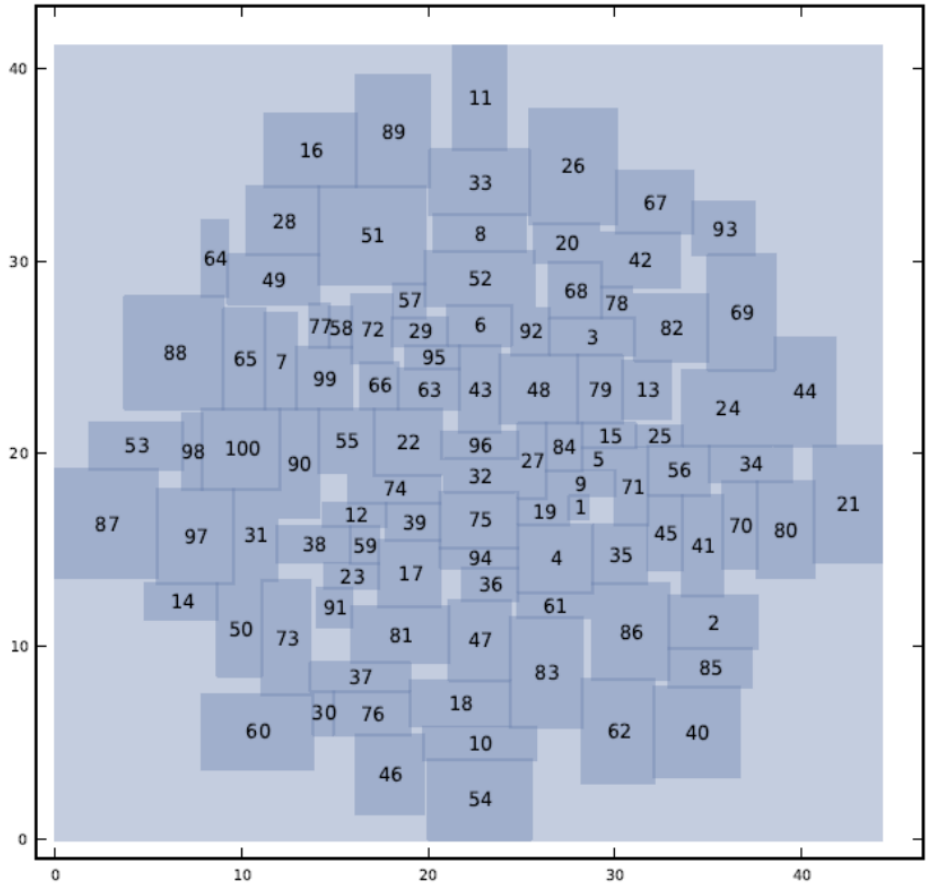
$$\max -(y-128)^2$$

$$y = x_0 + 2x_1 + 4x_2 + 8x_3 + 16x_4 + 32x_5 + 64x_6 + 128x_7$$
$$x_i \in \{0,1\}$$

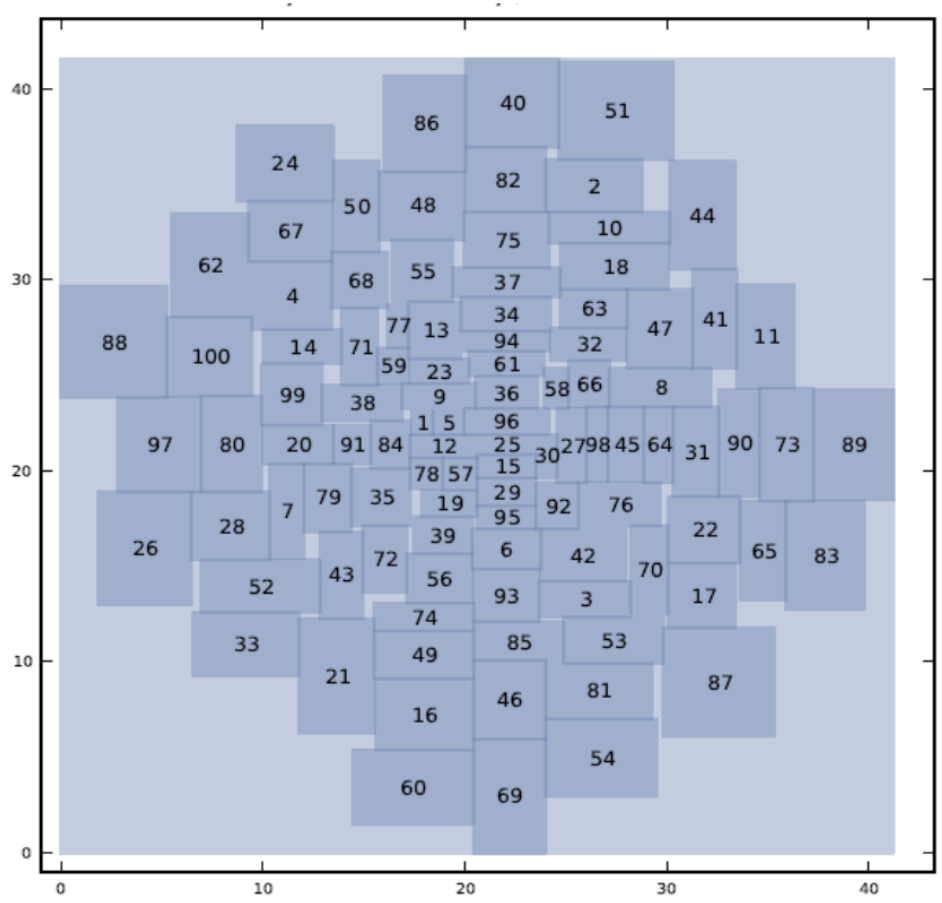
$(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (0,0,0,0,0,0,0,1)$ is optimal
What if start with the solution $(0,0,0,0,0,0,1,0)$? and you can only flip only one bit?

A biased random-key genetic algorithm for the unequal area facility layout problem

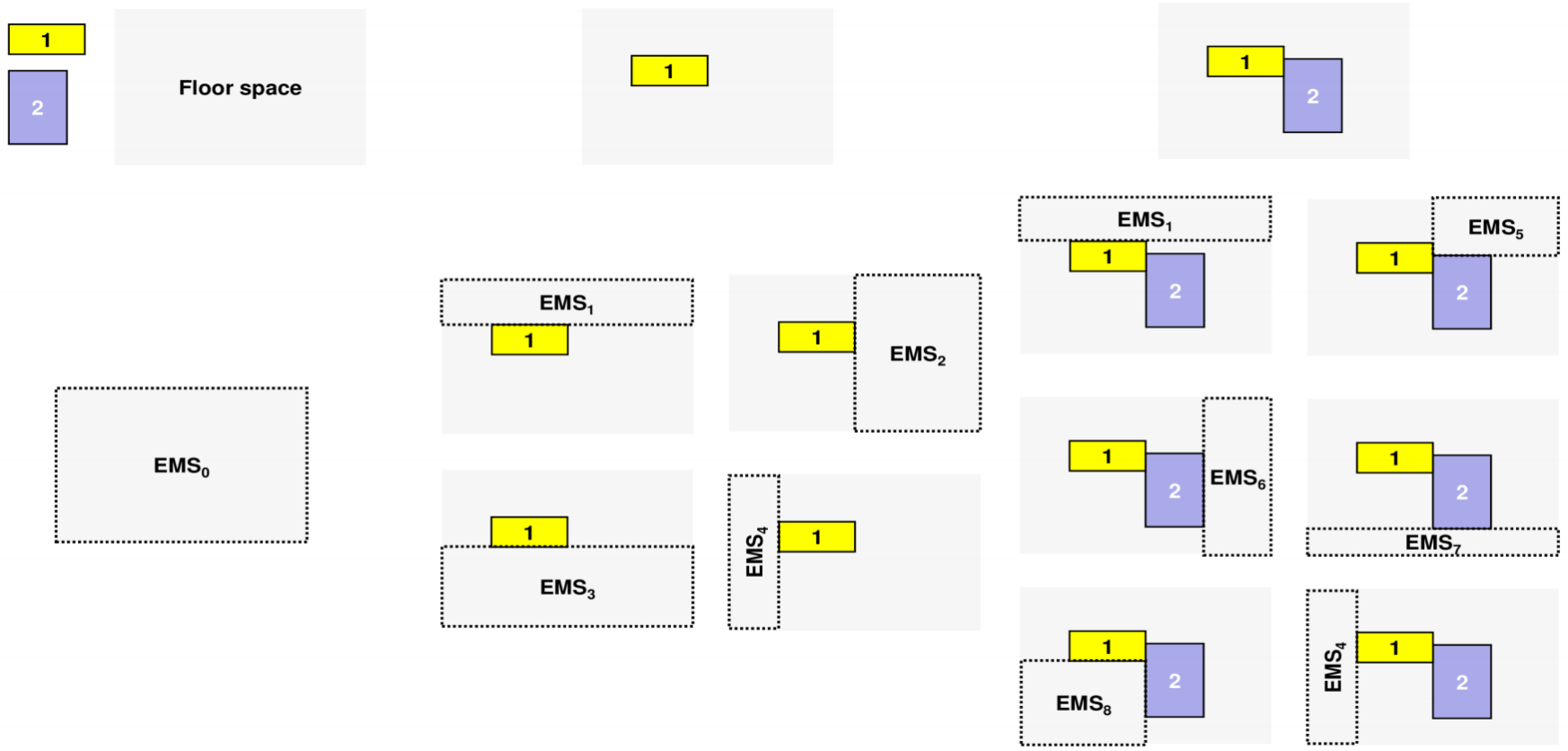
1st generation: 530404.76



50th generation: 478910.09



Transformer/Decoder – Find Empty Maximal Space



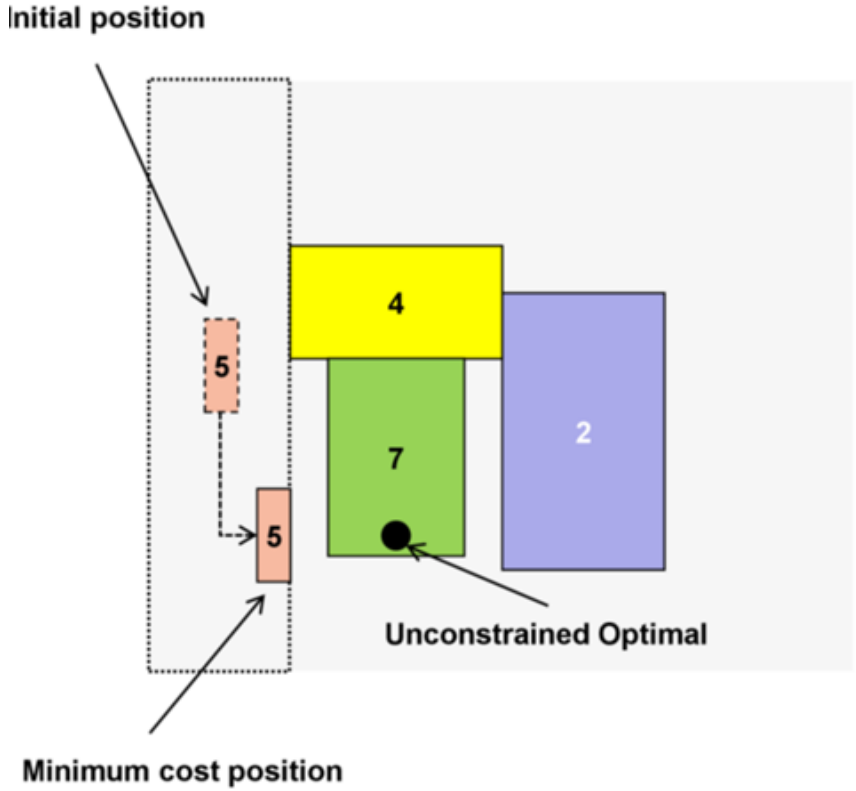
a) Facilities to be placed and the initial empty maximal-space (the floor space)

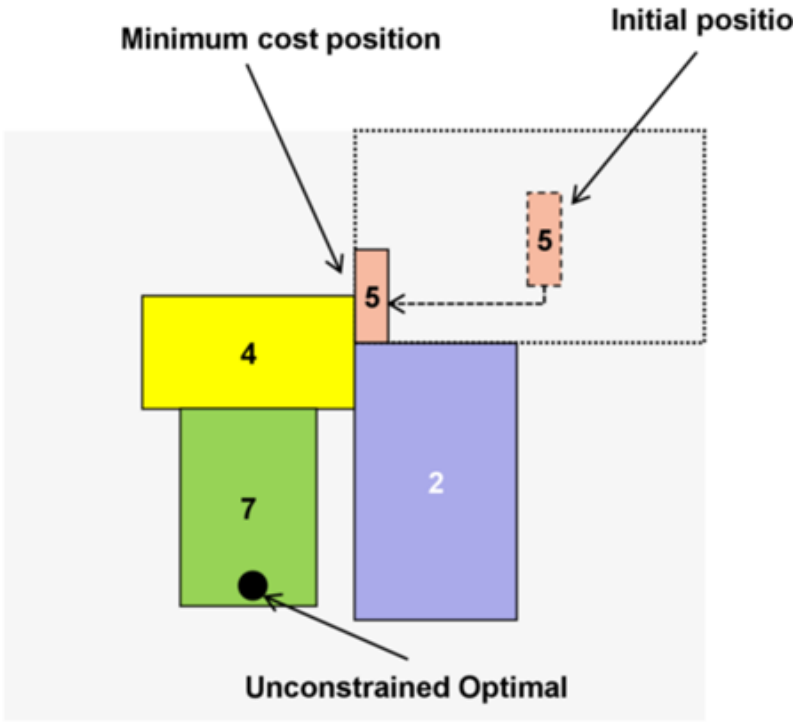
b) Empty maximal-spaces after placing facility 1.

c) Empty maximal-spaces after placing facility 2.

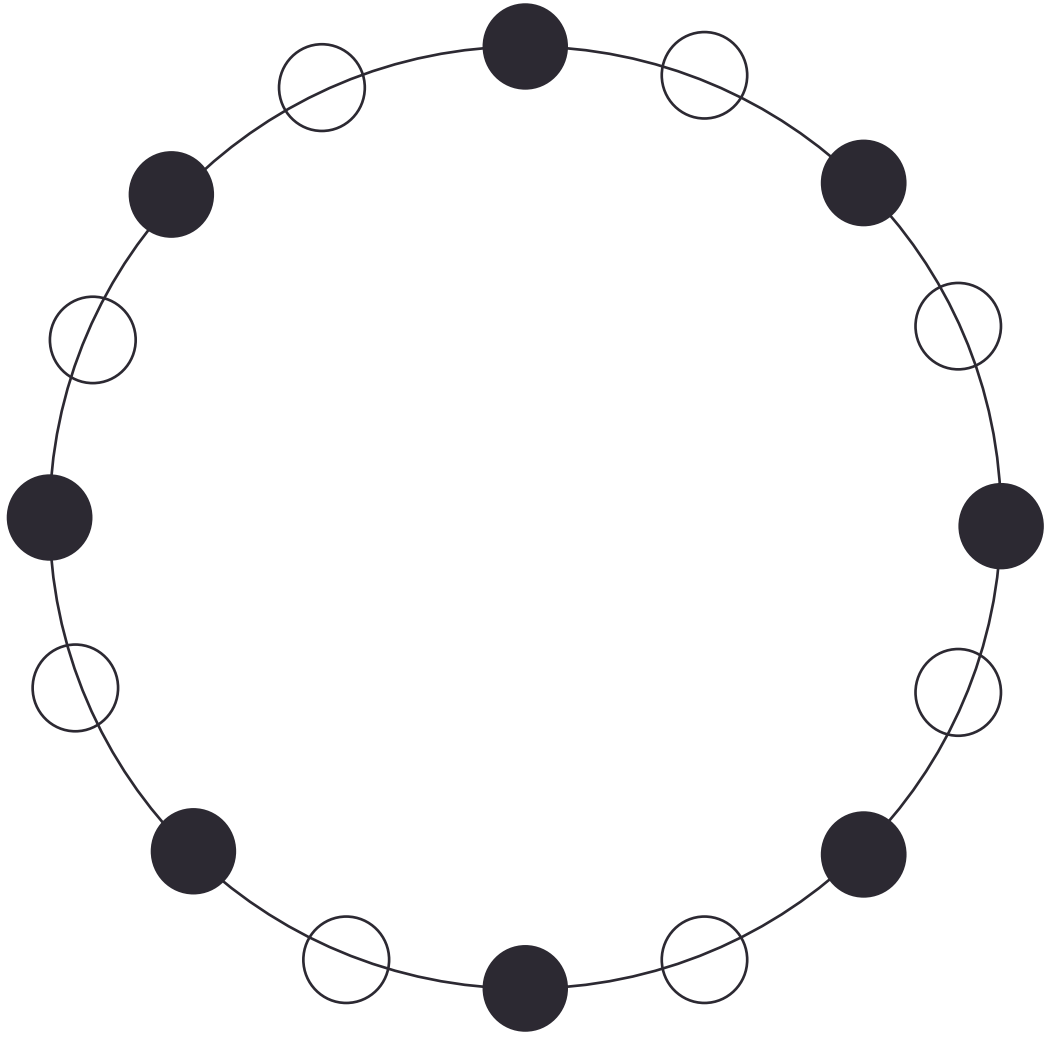
Transformer/Decoder – Space Representation

For each EMS in which the facility fits, we place the facility in the center of the EMS and move it as close as possible to the UO and compute the objective.

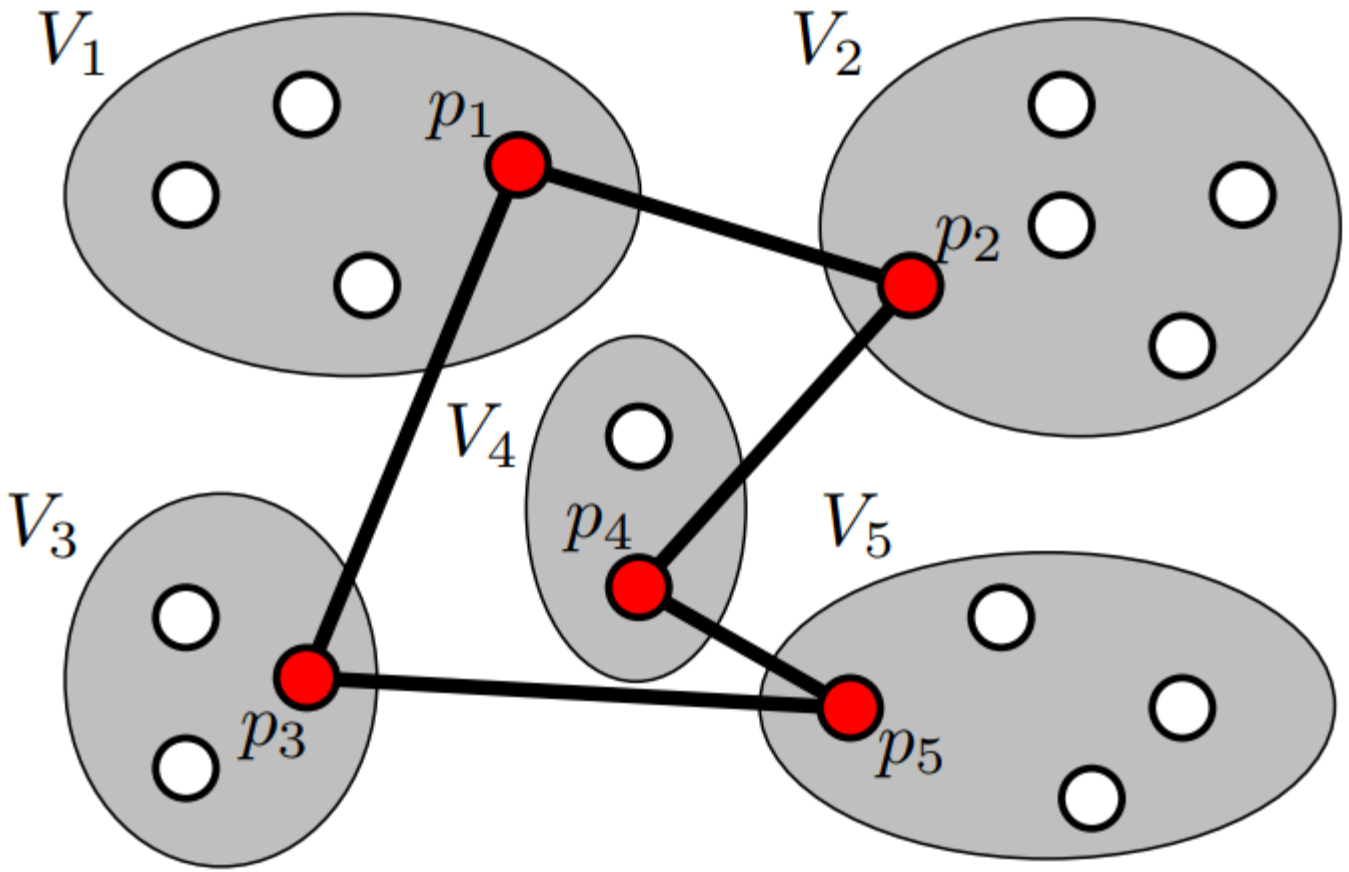




Traveling Salesman Problem – Delete and Min-Matching



Generalized TSP



Generalized TSP Representation - Sequencing

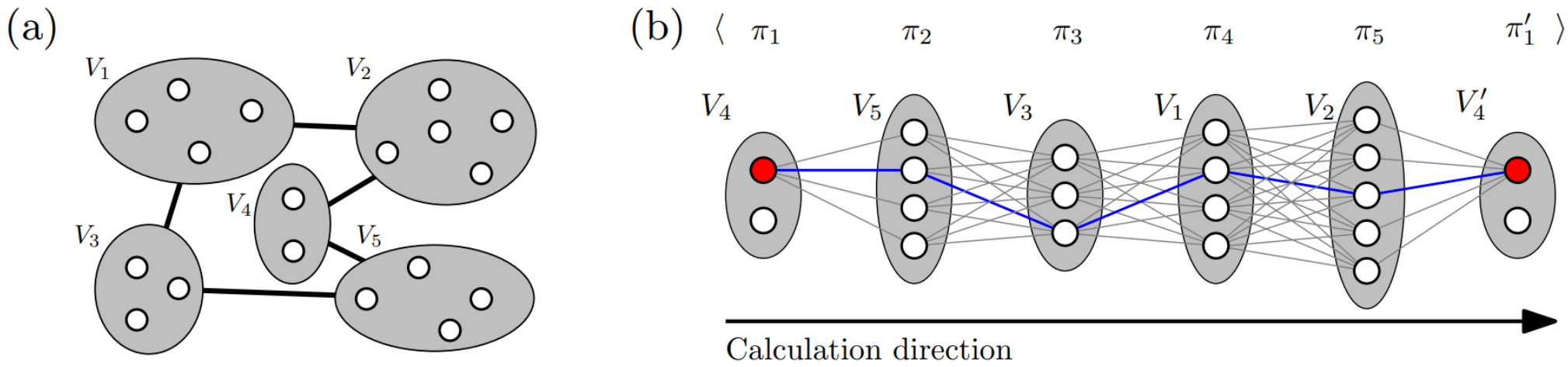


Fig. 2. (a) Visiting order of clusters characterized by permutation $\pi = \langle 4, 5, 3, 1, 2 \rangle$ and (b) corresponding graph on which the shortest path algorithm is applied, starting at the first node of cluster V_4 and ending at its clone in cluster V'_4.

Two fundamental classes of metaheuristics

1. Local search metaheuristics iteratively make changes to a single solution.
2. Population-based metaheuristics iteratively combine solutions into new ones.

Contents

1 Simulated Annealing (covered last week)

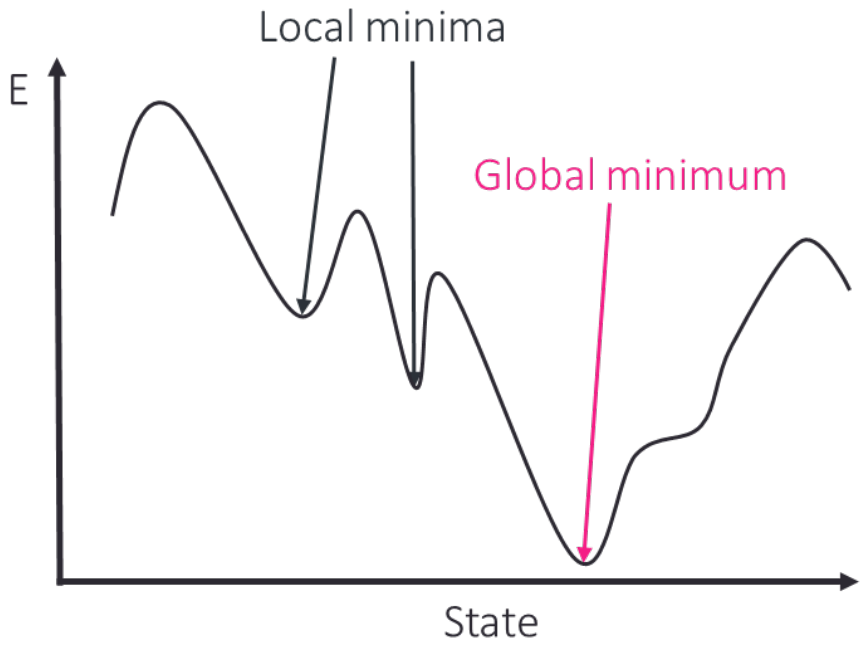
2 Tabu Search

Tabu Search

Tabu search, created by Fred W. Glover is a metaheuristic search method employing local search methods used for mathematical optimization.

Tabu search enhances the performance of local search by relaxing its basic rule.

- At each step, *worsening* moves can be accepted if no improving move is available (like when the search is stuck at a strict local minimum) .
- *Tabu* are introduced to discourage the search from coming back to previously-visited solutions.



*Fred Glover (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence". Computers and Operations Research. **13** (5): 533–549. doi:10.1016/0305-0548(86)90048-1.*

Tabu Search

Use memory to represent the visited solutions or user-defined sets of rules.

These memory structures form what is known as the Tabu List.

The strategy of using memory is that if a potential solution

- has already been visited within a specific short period of time
- has dishonored any defined rule

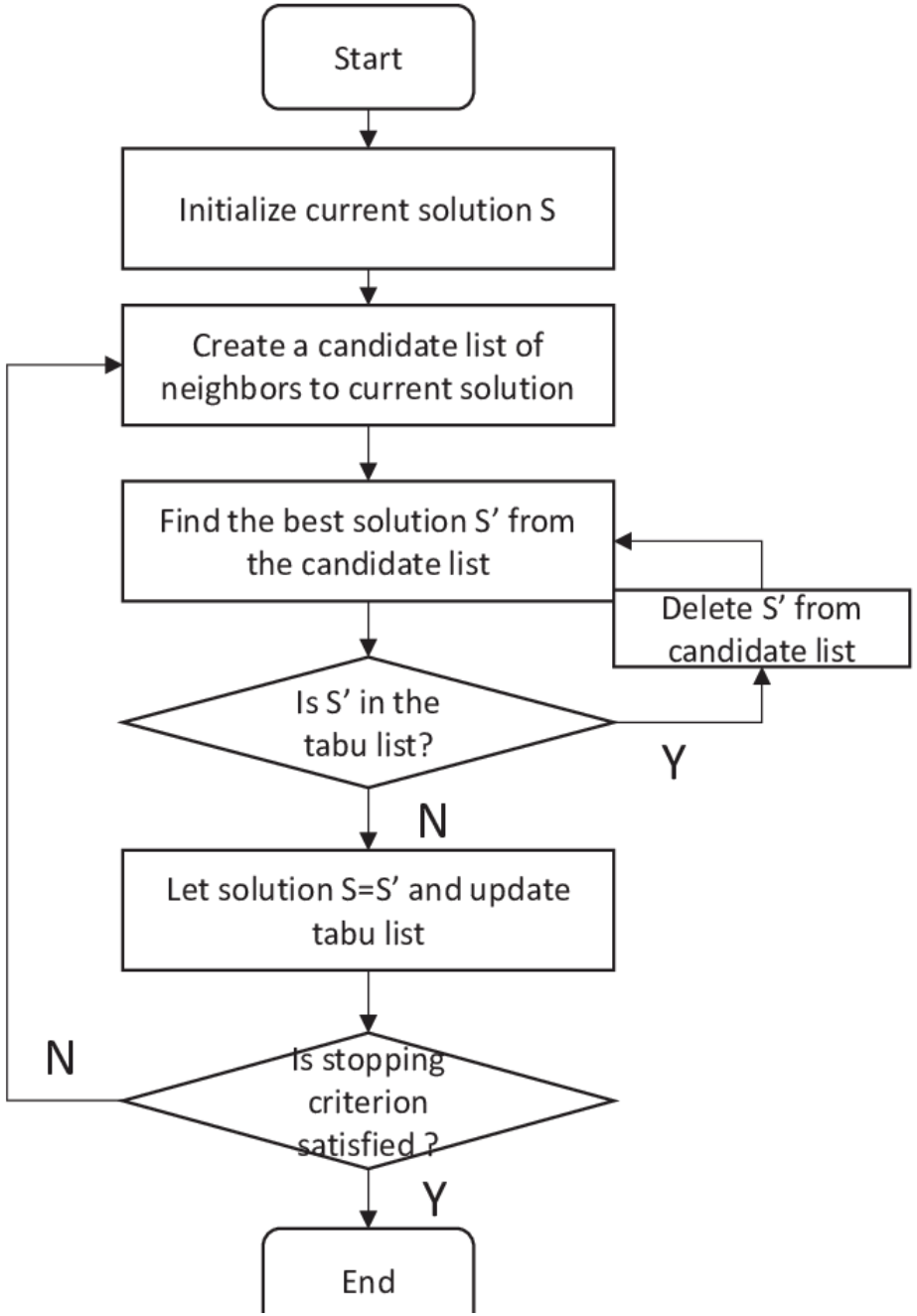
, it is labeled “taboo” so that the technique does not acknowledge that possibility repeatedly.

Tabu Search

Three memory structures:

- Short-term: This contains the list of solutions considered recently. If any potential solution arrives on this list, it cannot revisit until and unless it reaches an expiry point.
- Intermediate-term: A set of rules deliberately made to bias the exploration towards promising parts of the search space.
- Long-term: A set of rules that endorse assortment in the search procedure. It is basically represented by resets while the search get stuck in a plateau or reaches a suboptimal dead end.

Tabu Search


Start
Initialize current solution S
Create a candidate list of neighbors to current solution
Find the best solution S' from the candidate list
Delete S' from candidate list
Is S' in the tabu list?
Y
N
Let solution S=S' and update tabu list
N
Is stopping criterion satisfied ?
Y
End

Tabu Search——Example 1 TSP

(1) Taboo solutions that already been visited within a specific short period of time

Iteration	Current solution	2-opt
1	(1,2,3,4,5,6)	
2		
3		
4		
5		
6		
7		
…		

Tabu tenue=4

	Tabu list
1	(1,2,3,4,5,6)
2	
3	
4	
5	
6	
…	

Tabu Search——Example 1 TSP

(1) Taboo solutions that already been visited within a specific short period of time

Iteration	Current solution	2-opt
1	(1,2,3,4,5,6)	(2,5)
2	(1,5,3,4,2,6)	
3		
4		
5		
6		
7		
…		

Tabu tenue=4

	Tabu list
1	(1,2,3,4,5,6)
2	(1,5,3,4,2,6)
3	
4	
5	
6	
…	

Tabu Search——Example 1 TSP

(1) Taboo solutions that already been visited within a specific short period of time

Iteration	Current solution	2-opt
1	(1,2,3,4,5,6)	(2,5)
2	(1,5,3,4,2,6)	(5,3)
3	(1,3,5,4,2,6)	
4		
5		
6		
7		
…		

Tabu tenue=4

	Tabu list
1	(1,2,3,4,5,6)
2	(1,5,3,4,2,6)
3	(1,3,5,4,2,6)
4	
5	
6	
…	

Tabu Search——Example 1 TSP

(1) Taboo solutions that already been visited within a specific short period of time

Iteration	Current solution	2-opt
1	(1,2,3,4,5,6)	(2,5)
2	(1,5,3,4,2,6)	(5,3)
3	(1,3,5,4,2,6)	(4,1)
4	(4,3,5,1,2,6)	
5		
6		
7		
…		

Tabu tenue=4

	Tabu list
1	(1,2,3,4,5,6)
2	(1,5,3,4,2,6)
3	(1,3,5,4,2,6)
4	(4,3,5,1,2,6)
5	
6	
…	

Tabu Search——Example 1 TSP

(1) Taboo solutions that already been visited within a specific short period of time

Tabu tenue=4

Iteration	Current solution	2-opt
1	(1,2,3,4,5,6)	(2,5)
2	(1,5,3,4,2,6)	(5,3)
3	(1,3,5,4,2,6)	(4,1)
4	(4,3,5,1,2,6)	(1,4)
5	(4,3,5,1,2,6)	
6		
7		
…		

	Tabu list
1	(1,2,3,4,5,6)
2	(1,5,3,4,2,6)
3	(1,3,5,4,2,6)
4	(4,3,5,1,2,6)
5	
6	
…	

Tabu Search——Example 1 TSP

(1) Taboo solutions that already been visited within a specific short period of time

Iteration	Current solution	2-opt
1	(1,2,3,4,5,6)	(2,5)
2	(1,5,3,4,2,6)	(5,3)
3	(1,3,5,4,2,6)	(4,1)
4	(4,3,5,1,2,6)	(1,4)
5	(4,3,5,1,2,6)	(6,3)
6	(4,6,5,1,2,3)	
7		
…		

Tabu tenue=4

	Tabu list
~~1~~	~~(1,2,3,4,5,6)~~
2	(1,5,3,4,2,6)
3	(1,3,5,4,2,6)
4	(4,3,5,1,2,6)
5	(4,6,5,1,2,3)
6	
…	

Tabu Search——Example 1 TSP

(1) Taboo solutions that already been visited within a specific short period of time

Iteration	Current solution	2-opt
1	(1,2,3,4,5,6)	(2,5)
2	(1,5,3,4,2,6)	(5,3)
3	(1,3,5,4,2,6)	(4,1)
4	(4,3,5,1,2,6)	(1,4)
5	(4,3,5,1,2,6)	(6,3)
6	(4,6,5,1,2,3)	(4,5)
7	(5,6,4,1,2,3)	
…		

Tabu tenue=4

	Tabu list
~~1~~	~~(1,2,3,4,5,6)~~
~~2~~	~~(1,5,3,4,2,6)~~
3	(1,3,5,4,2,6)
4	(4,3,5,1,2,6)
5	(4,6,5,1,2,3)
6	(5,6,4,1,2,3)
…	

Tabu Search——Example 2 TSP

(2) Taboo 2-opt operators

Iteration	Current solution	2-opt
1	(1,2,3,4,5,6)	
2		
3		
4		
5		
6		
7		
…		

Tabu tenue=4

	Tabu list
1	
2	
3	
4	
5	
6	
…	

Tabu Search——Example 2 TSP

(2) Taboo 2-opt operators

Iteration	Current solution	2-opt
1	(1,2,3,4,5,6)	(1,4)
2	(4,2,3,1,5,6)	
3		
4		
5		
6		
7		
…		

Tabu tenue=4

	Tabu list
1	(1,4)
2	
3	
4	
5	
6	
…	

Tabu Search——Example 2 TSP

(2) Taboo 2-opt operators

Iteration	Current solution	2-opt
1	(1,2,3,4,5,6)	(1,4)
2	(4,2,3,1,5,6)	(4,5)
3	(5,2,3,1,4,6)	
4		
5		
6		
7		
…		

Tabu tenue=4

	Tabu list
1	(1,4)
2	(4,5)
3	
4	
5	
6	
…	

Tabu Search——Example 2 TSP

(2) Taboo 2-opt operators

Iteration	Current solution	2-opt
1	(1,2,3,4,5,6)	(1,4)
2	(4,2,3,1,5,6)	(4,5)
3	(5,2,3,1,4,6)	(5,1)
4	(1,2,3,5,4,6)	
5		
6		
7		
…		

Tabu tenue=4

	Tabu list
1	(1,4)
2	(4,5)
3	(5,1)
4	
5	
6	
…	

Tabu Search——Example 2 TSP

(2) Taboo 2-opt operators

Iteration	Current solution	2-opt
1	(1,2,3,4,5,6)	(1,4)
2	(4,2,3,1,5,6)	(4,5)
3	(5,2,3,1,4,6)	(5,1)
4	(1,2,3,5,4,6)	(3,2)
5	(1,3,2,5,4,6)	
6		
7		
…		

Tabu tenue=4

	Tabu list
1	(1,4)
2	(4,5)
3	(5,1)
4	(3,2)
5	
6	
…	

Tabu Search——Example 2 TSP

(2) Taboo 2-opt operators

Iteration	Current solution	2-opt
1	(1,2,3,4,5,6)	(1,4)
2	(4,2,3,1,5,6)	(4,5)
3	(5,2,3,1,4,6)	(5,1)
4	(1,2,3,5,4,6)	(3,2)
5	(1,3,2,5,4,6)	(6,2)
6	(1,3,6,5,4,2)	
7		
…		

Tabu tenue=4

	Tabu list
~~1~~	~~(1,4)~~
2	(4,5)
3	(5,1)
4	(3,2)
5	(6,2)
6	
…	

Tabu Search——Example 2 TSP

(2) Taboo 2-opt operators

Iteration	Current solution	2-opt
1	(1,2,3,4,5,6)	(1,4)
2	(4,2,3,1,5,6)	(4,5)
3	(5,2,3,1,4,6)	(5,1)
4	(1,2,3,5,4,6)	(3,2)
5	(1,3,2,5,4,6)	(6,2)
6	(1,3,6,5,4,2)	(3,2)
7	(1,3,6,5,4,2)	(1,4)
…		

	Tabu list
~~1~~	~~(1,4)~~
~~2~~	~~(4,5)~~
3	(5,1)
4	(3,2)
5	(6,2)
6	(1,4)
…	

3 Variable Neighborhood Search

Variable Neighborhood Search

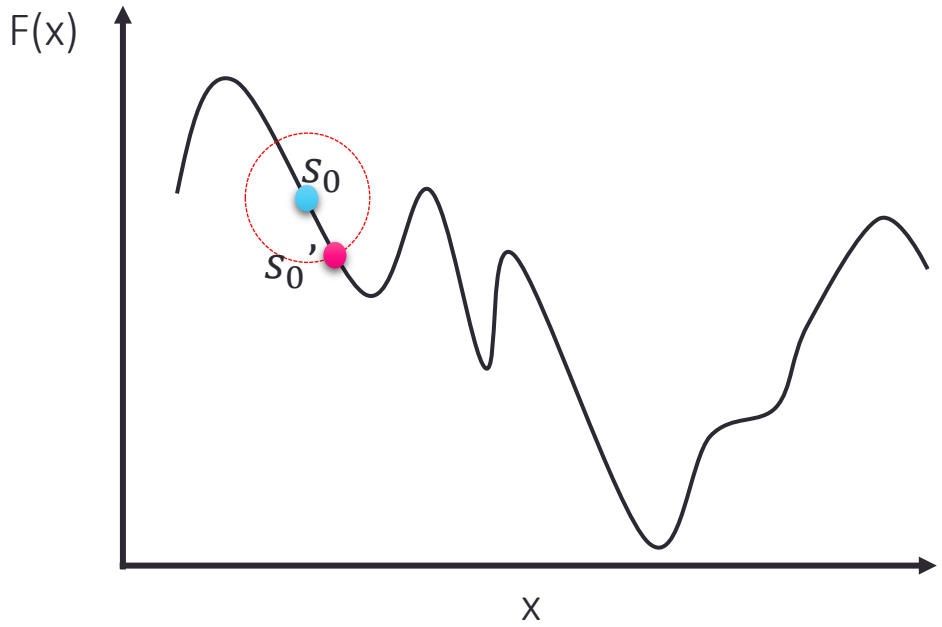
- Variable Neighborhood Search (VNS) is a metaheuristic based upon a simple principle: systematic change of neighborhood within the search.
- At the initialization step, a set of neighborhood structure has to be defined. These neighborhoods can be arbitrarily chosen.
- Then an initial solution is generated, and the main cycle of VNS begins. However, such a sequence may produce an in efficient search, because a large number of solutions can be revisited.
- So, generate another solution and explore its neighborhood.

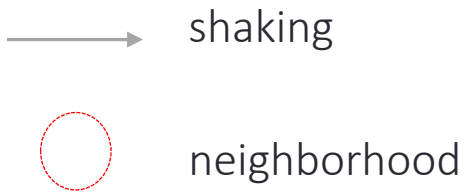
Variable Neighborhood Search

- This cycle consists of three steps: shaking, local search and move.
- In the shaking step, a solution s_0 is randomly selected in the nth neighborhood of the current solution s.
- Then, s_0 is used as the initial solution of a local search procedure, to generate the solution s_0'. The local search can use any neighborhood structure.
- At the end of the local search process, if s_0'is better than s_0 , then s_0' replaces s and the cycle starts again with n = 1.
- Otherwise, the algorithm moves to the next neighborhood n + 1 and a new shaking phase starts using this neighborhood.

Variable Neighborhood Search

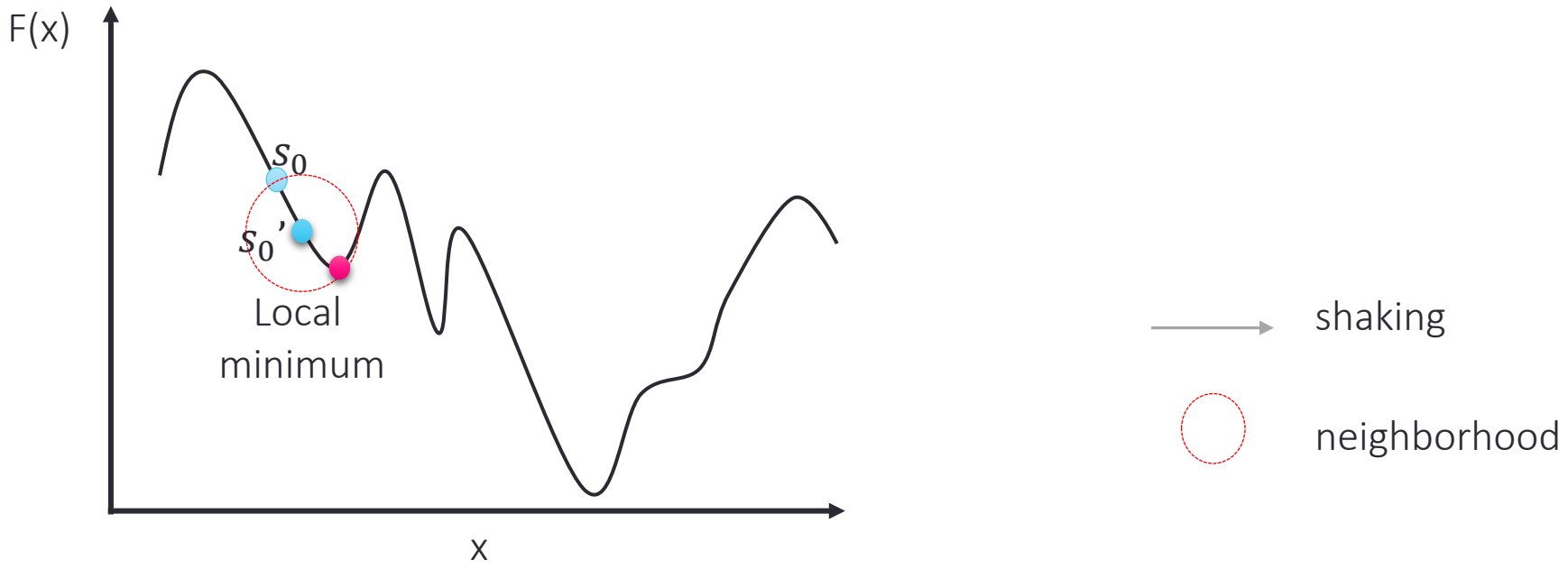
- Initial a solution s_0
- search its neighborhood to find s_0' is better than s_0





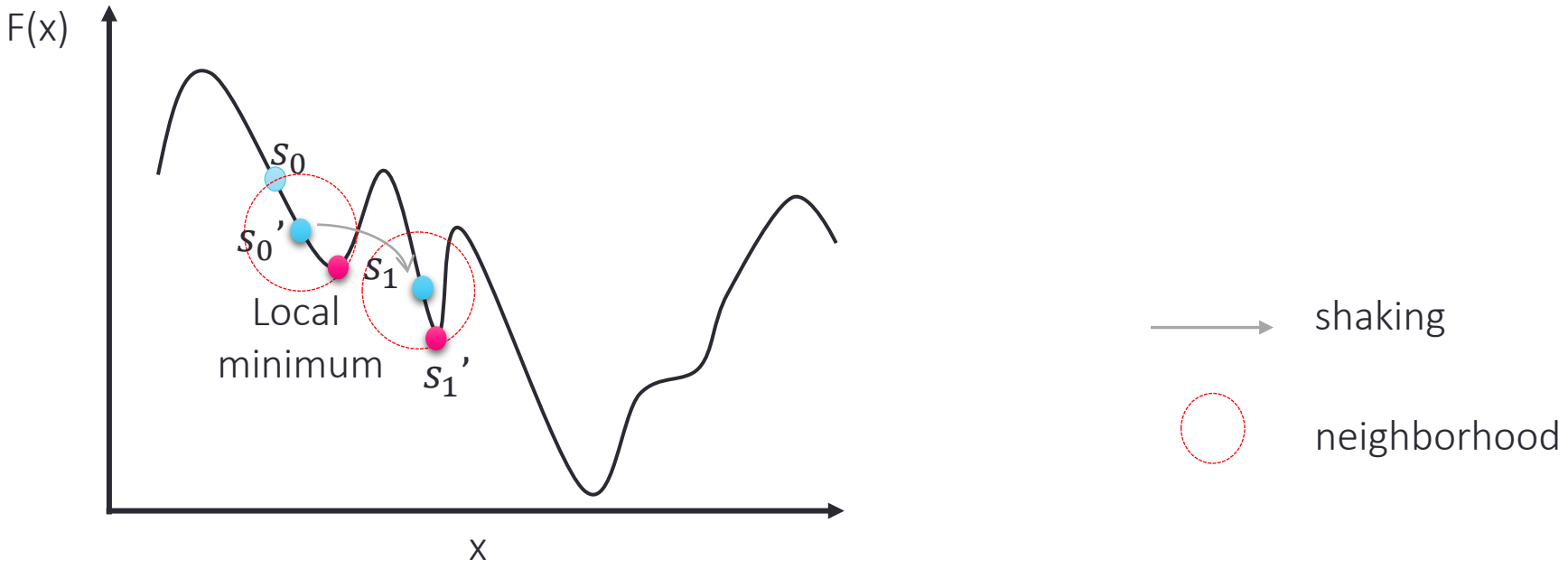
Variable Neighborhood Search

- Search the neighbor of s_0' , and find a local minimum.



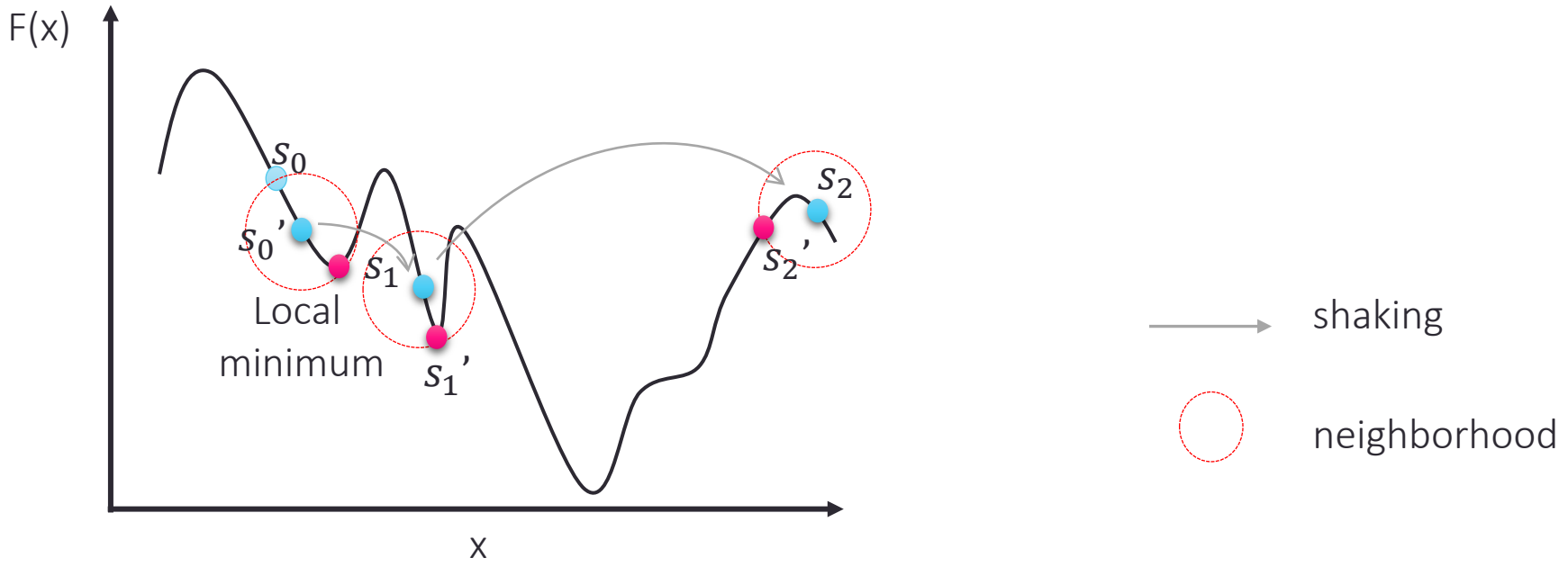
Variable Neighborhood Search

- Shake: initial a solution s_1
- Local search its neighborhood to find s_1’ , and reach a local optimal



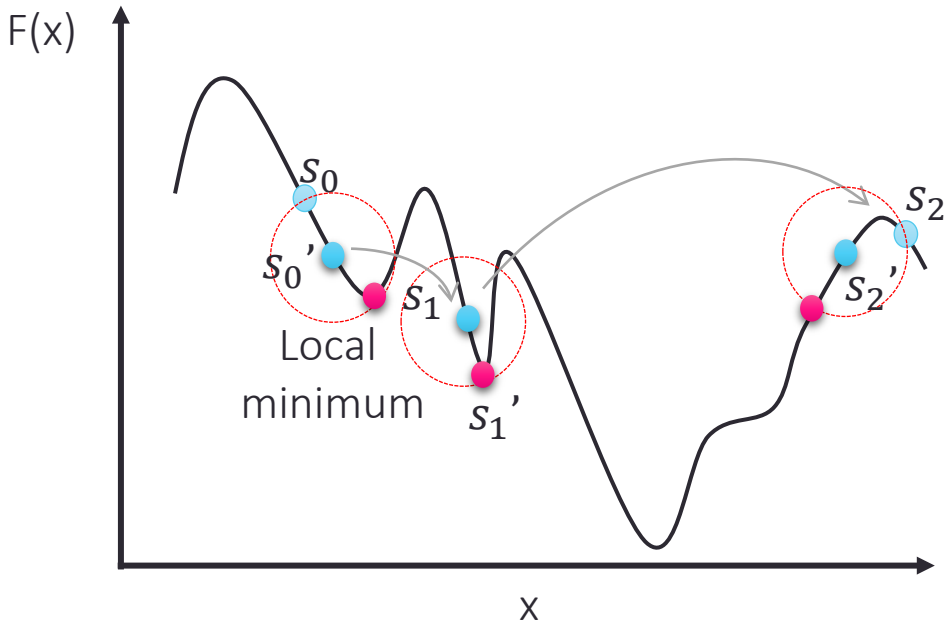
Variable Neighborhood Search

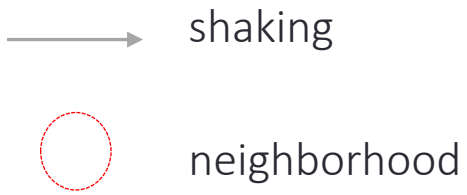
- Shake: initial a solution s_2
- Local search its neighborhood to find s_2’ , and reach a local optimal



Variable Neighborhood Search

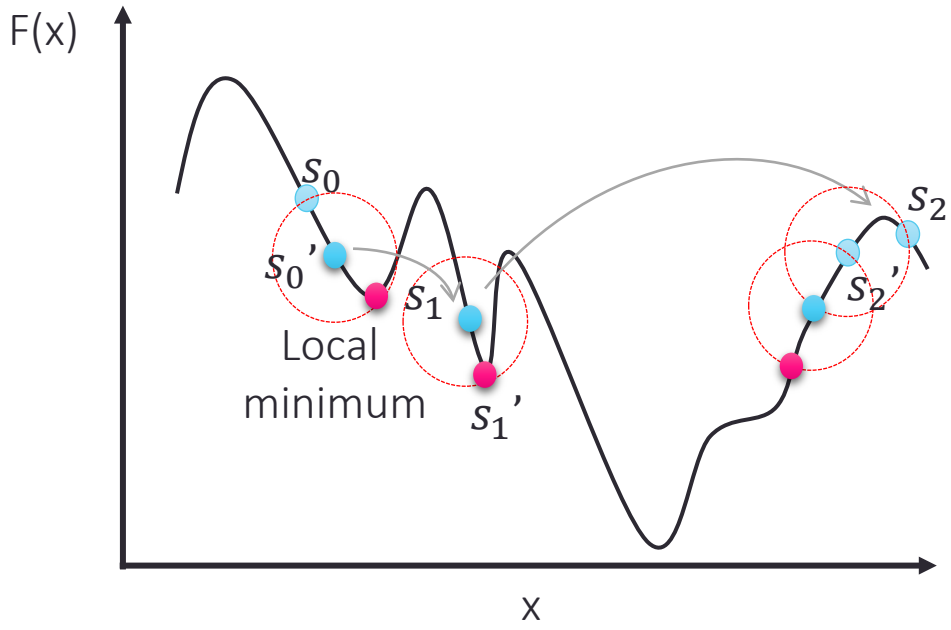
- Local search the neighbor of s_2'





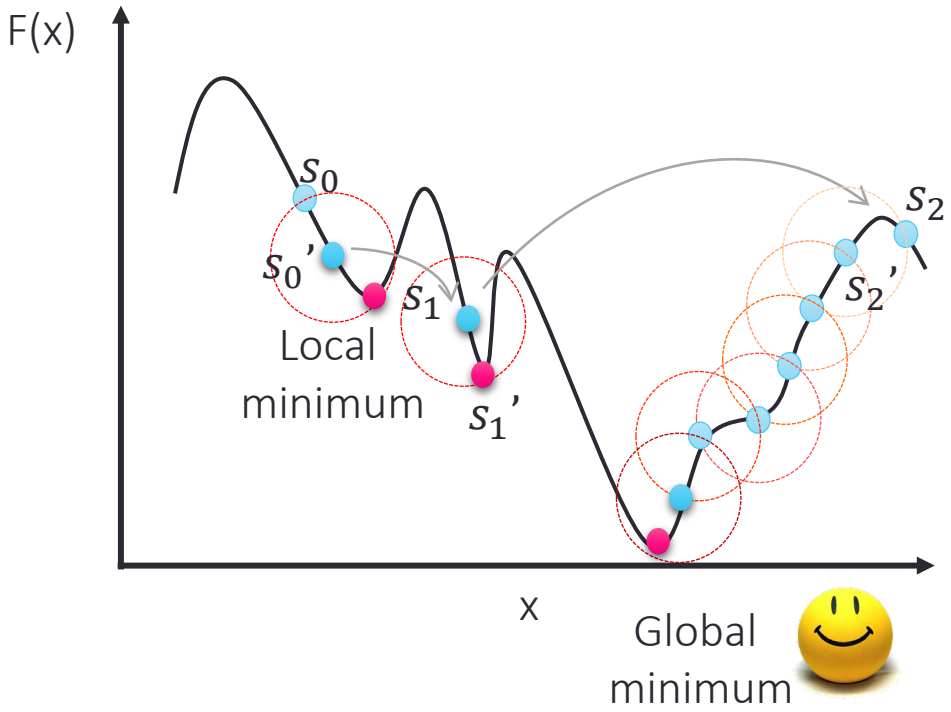
Variable Neighborhood Search

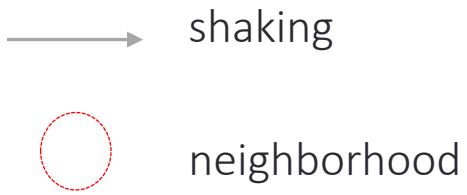
- Continue local search, and shaking



Variable Neighborhood Search

- Until find the global optimal





Variable Neighborhood Search

VNS

1 Select a set of neighborhood structures N_n, $n = 1, ..., n_{max}$
2 Choose, at random, an initial solution s in the search space
3 **while** the stopping criterion is not satisfied **do**
4 $n \leftarrow 1$
5 **while** $n \leq n_{max}$ **do**
6 Shaking: select a random solution s' in the n^{th} neighborhood $N_n(s)$ of s
7 Apply a local search starting from s' to get a solution s''
8 **if** s'' is better than s **then**
9 $s \leftarrow s''$
10 $n \leftarrow 1$
11 **else**
12 $n \leftarrow n + 1$
13 **end**
14 **end**
15 **end**
16 **return** the best solution met

Variable Neighborhood Descent (VND)

VND

1 Select a set of neighborhood structures $N_n, n = 1, ..., n_{max}$
2 Choose, at random, an initial solution s in the search space
3 $n \leftarrow 1$
4 **while** $n < n_{max}$ **do**
5 Select the best solution s' in the n^{th} neighborhood $N_n(s)$ of s
6 **if** s' is better than s **then**
7 $s \leftarrow s'$
8 $n \leftarrow 1$
9 **else**
10 $n \leftarrow n + 1$
11 **end**
12 **end**
13 **return** the best solution met

4 Iterated Local Search

Iterated Local Search

- ILS is a metaheuristic based on a simple idea: instead of repeatedly applying a local search procedure to randomly generated starting solutions, ILS generates the starting solution for the next iteration by perturbing the local optimum found at the current iteration.
- This is done in the expectation that the perturbation mechanism provides a solution located in the basin of attraction of a better local optimum.

Iterated Local Search

1. The perturbation mechanism is a key feature of ILS.

- a too weak perturbation may not be sufficient to escape from the basin of attraction of the current local optimum;
- a too strong perturbation would make the algorithm similar to multi-start local search with randomly generate d starting solutions.

2. The acceptance criterion defines the conditions that the new local optimum p^* has to satisfy in order to replace the current one s^*.

Iterated Local Search

The acceptance criterion, combined with the perturbation mechanism, enables controlling the trade-off between intensification and diversification.

- An extreme acceptance criterion in terms of intensification is to accept only improving solutions.
- Another extreme criterion in terms of diversification is to accept any solution, without regard to its quality.

Iterated Local Search

ILS

```
1  Choose, at random, an initial solution s in the search space
2  Apply a local search starting from s to get a solution s*
3  repeat
4  |   Perturb s* to get a solution p
5  |   Apply a local search starting from p to get a solution p*
6  |   if the acceptance criterion is satisfied then
7  |   |   s* ← p*
8  |   end
9  until the stopping criterion is satisfied
10 return the best solution met
```

5 Large Neighborhood Search

Large Neighborhood Search

Most neighborhood search algorithms explicitly define the neighborhood like the relocate neighborhood described.

In the Large Neighborhood Search(LNS) metaheuristic the neighborhood is defined implicitly by a *destroy* and a *repair* method. A destroy method destructs part of the current solution while a repair method rebuilds the destroyed solution.

The neighborhood $N(x)$ of a solution x is then defined as the set of solutions that can be reached by first applying the destroy method and then the repair method.

Large Neighborhood Search

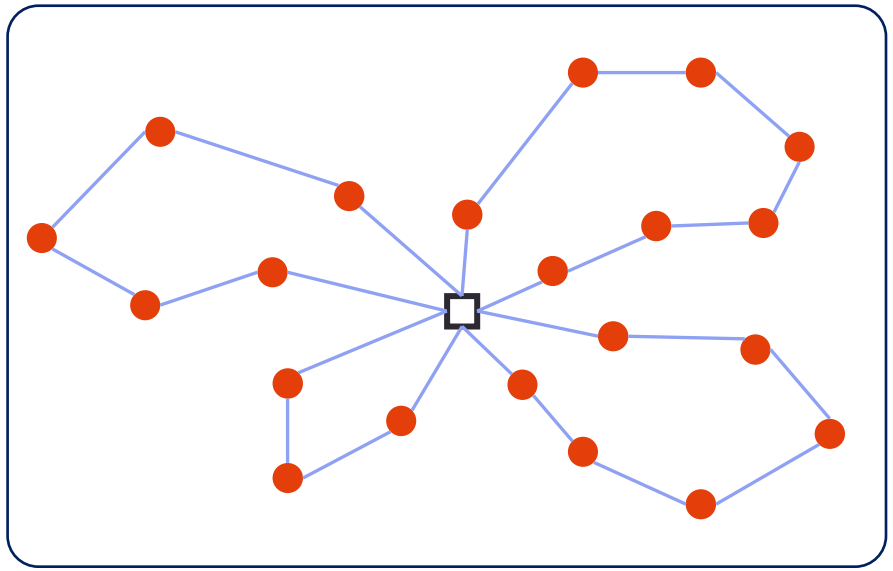
The function $d(\cdot)$ is the destroy method while $r(\cdot)$ is the repair method.

Algorithm 1 Large neighborhood search

```
1:  input: a feasible solution x
2:  x^b = x;
3:  repeat
4:      x^t = r(d(x));
5:      if accept(x^t, x) then
6:          x = x^t;
7:      end if
8:      if c(x^t) < c(x^b) then
9:          x^b = x^t;
10:     end if
11: until stop criterion is met
12: return x^b
```

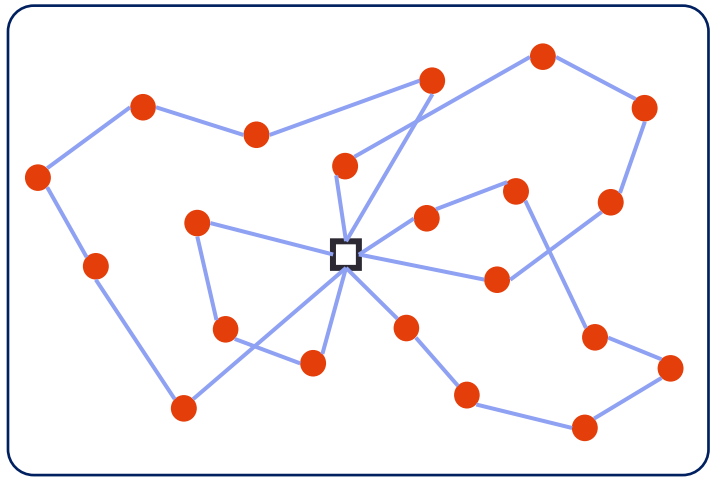
Large Neighborhood Search——Example (CVRP)

The *capacitated vehicle routing problem* (CVRP) is a VRP in which vehicles with limited carrying capacity need to pick up or deliver items at various locations. The items have a quantity, such as weight or volume, and the vehicles have a maximum *capacity* that they can carry. The problem is to pick up or deliver the items for the least cost, while never exceeding the capacity of the vehicles.

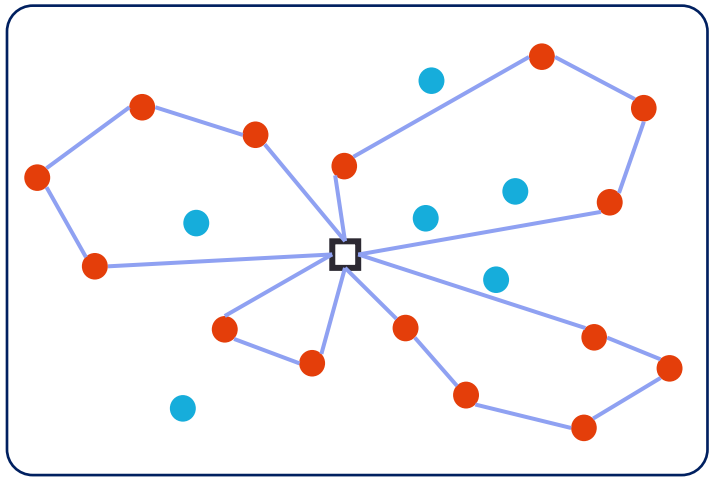


Large Neighborhood Search——Example（CVRP)

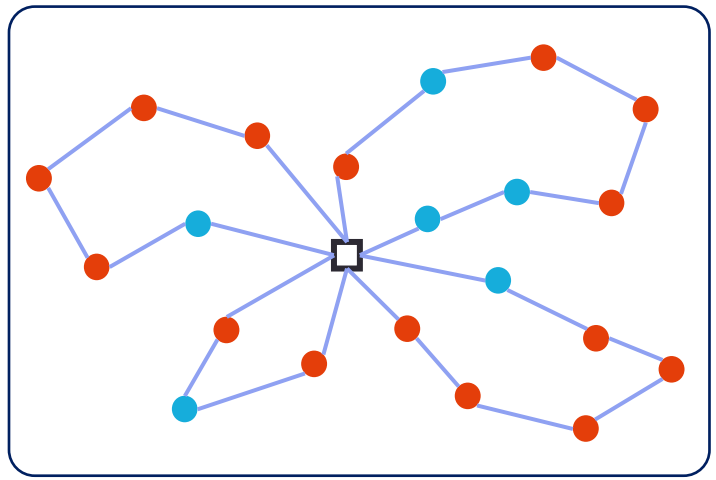
Destroy and repair a CVRP solution



a CVRP solution before the destroy operation



the solution after a destroy operation that removed 6 customers



the solution after the repair operation has reinserted the customers

Adaptive large neighborhood search

- *The Adaptive Large Neighborhood Search* (ALNS) heuristic extends the LNS heuristic by allowing multiple destroy and repair methods to be used within the same search.
- Each destroy/repair method is assigned a weight that controls how often the particular method is attempted during the search.
- The weights are adjusted dynamically as the search progresses so that the heuristic adapts to the instance at hand and to the state of the search.

Adaptive large neighborhood search

Algorithm 2 Adaptive large neighborhood search

1: input: a feasible solution x
2: $x^b = x$; $\rho^- = (1, \ldots, 1)$; $\rho^+ = (1, \ldots, 1)$;
3: **repeat**
4: select destroy and repair methods $d \in \Omega^-$ and $r \in \Omega^+$ using ρ^- and ρ^+;
5: $x^t = r(d(x))$;
6: **if** accept(x^t, x) **then**
7: $x = x^t$;
8: **end if**
9: **if** $c(x^t) < c(x^b)$ **then**
10: $x^b = x^t$;
11: **end if**
12: update ρ^- and ρ^+;
13: **until** stop criterion is met
14: **return** x^b

Reference

1. *Gendreau, M., & Potvin, J. Y. (Eds.). (2010). Handbook of metaheuristics (Vol. 2, p. 9). New York: Springer.*
2. Boussaïd, I., Lepagnot, J., & Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences*, *237*, 82–117. https://doi.org/10.1016/j.ins.2013.02.041