Lecture 6 - Evolutionary Combinatorial Optimisation

Yuhui Shi

CSE, SUSTech

Review of Previous Lectures

- Self-adaptation and Parameter Control in Evolutionary Algorithms
- Representation is Important
- Popular Evolutionary Algorithms Variants: DE, PSO, ES, BSO, etc.

Outline of This Lecture

- Brief Introduction to Combinatorial Optimisation
- Travelling Salesman Problem
- Cutting Stock Problem
- Summary

Outline of This Lecture

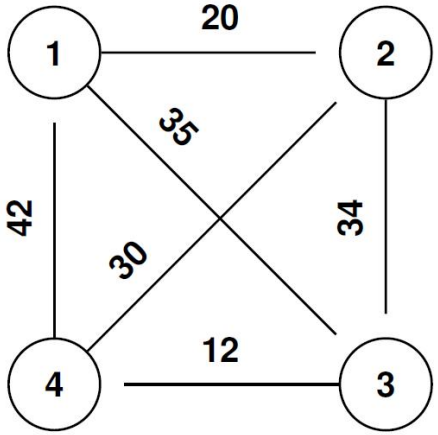
- **Brief Introduction to Combinatorial Optimisation**
- Travelling Salesman Problem
- Cutting Stock Problem
- Summary

Combinatorial Optimisation (CO, 组合优化)

- Finding an optimal solution from a finite set of feasible solutions in a discrete space.
- Methods: exact or approximation algorithms.
- Types of CO: routing, scheduling, permutation, assignment.

Permutation-based Optimisation Problems

- A class of CO problems.
- Solutions are represented as permutations (排列).
- Example:



Assumie starting from city 1:

Permutation 1	**1**	**2**	**3**	**4**	**1**
Permutation 2	**1**	**3**	**2**	**4**	**1**

. . . . . .

. . . . . .

Permutation-based Optimisation Problems *-Applications*

- Travelling Salesman Problem (TSP).
- Vehicle Routing Problem (VRP).
- Cutting Stock Problem (CSP).
- Time Tabling Problem.
- ...

Representation Issues in CO

- Unlike numerical function optimisation, a proper representation of combinatorial structures can be very difficult to find.
- A good representation should
  - ✓facilitate the smooth evolution towards global optima (e.g., avoid unnecessary local optima, etc.) and
  - ✓increase the probability of producing better offspring.
- Crossover can be particularly tricky to design, because we want offspring to inherit important structural information from parent.
- Mutation might also be tricky to design because infeasible offspring might occur.

Outline of This Lecture

- Brief Introduction to Combinatorial Optimisation
- **Travelling Salesman Problem**
- Cutting Stock Problem
- Summary

Travelling Salesman Problem (TSP) [1]

- 旅行推销员问题/最短回路问题
- Aim: find a shortest tour of *n* cities by visiting each city exactly once and returning to the starting city.

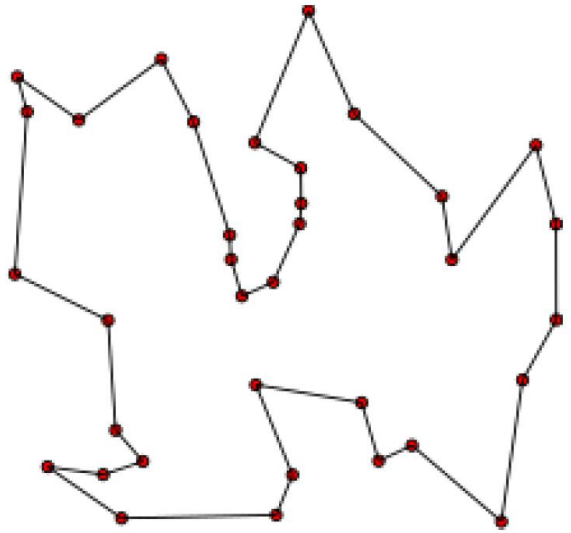


Figure 1: A solution to a TSP problem (figure from Wikipedia).

Formalisation of TSP

Given

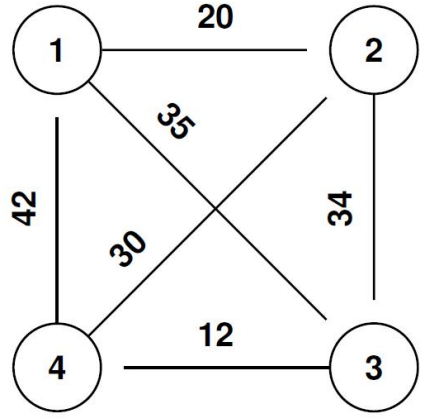
- n cities and
- $d_{i,j}$, the cost/distance of city i to city j, $\forall i, j \in \{1, \ldots, n\}$,

among them, find a permutation $(x_1, x_2, \ldots, x_n)$ of $(1, 2, \ldots, n)$ such that

$$D = \sum_{i=1}^{n} d_{x_i, x_{i+1}}$$

is minimised, where $x_{n+1} = x_1$.

Distance Matrix
-*Example*

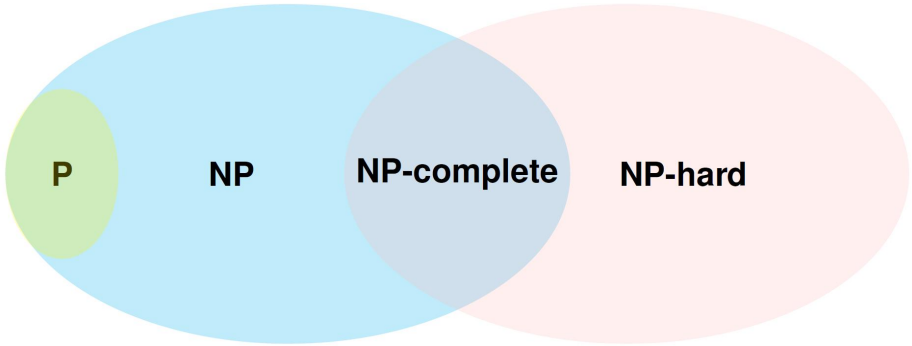


	1	2	3	4
1	-	20	35	42
2	20	-	34	30
3	35	34	-	12
4	42	30	12	-

Example: $d_{1,2} = 20$, the cost/distance of city 1 to city 2.

P, NP, NP-hard and NP-complete

- P-problem: A problem that the number of steps needed to solve it is bounded by a polynomial in the problem's size.
- NP-problem: The problem is solvable in polynomial time by a nondeterministic Turing machine. A P-problem is always also NP.
- NP-hard: ... if an algorithm for solving it can be reduced into one for solving any other NP-problem.
- NP-complete: Class of decision problems which contains the hardest problems in NP.



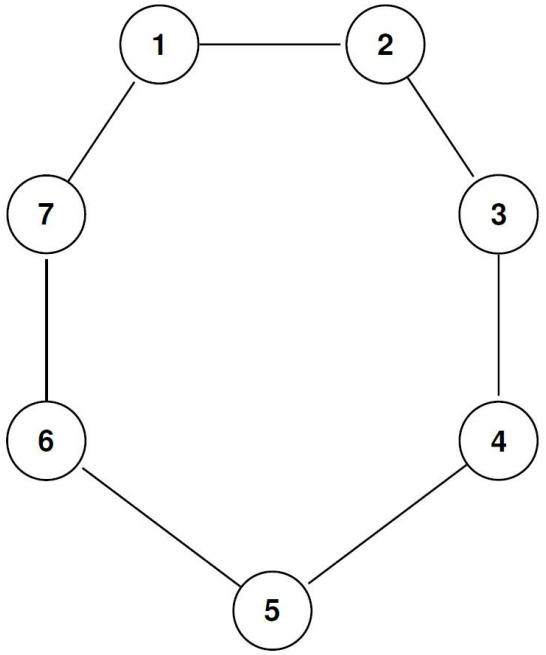
Solving TSP is Hard

- Finding an exact solution is NP-hard ← heuristics.

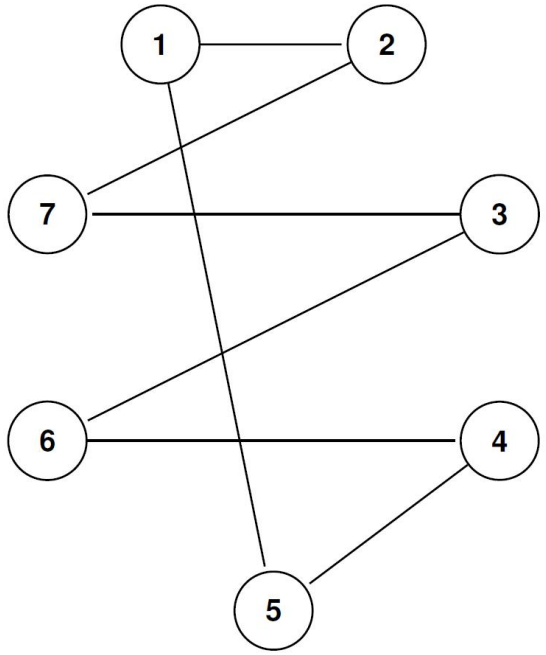
  NP-hard is not “NOT SOLVABLE”, you can not solve it efficiently if NP!=P.

Tour Representation
-*Example*

- $(x_1, x_2, \ldots, x_n)$ means that starting from city x_1, visiting x_2, then x_3, then . . . , then x_n and finally going back to x_1.



Example 1: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$



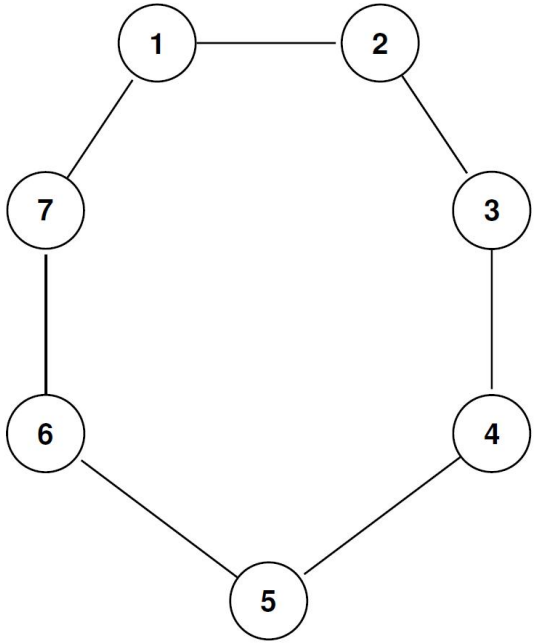
Example 2: $(x_1, x_2, x_7, x_3, x_6, x_4, x_5)$

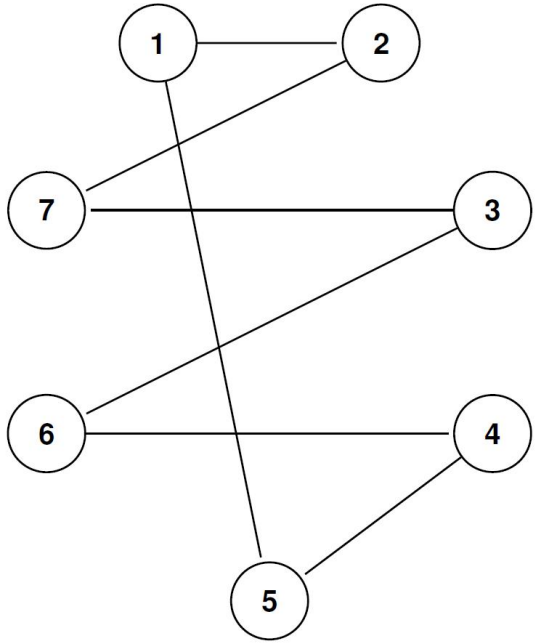
Simple Recombination (Crossover)

1. Construct an edge map from 2 parental tours.
2. Construct a child tour from the edge map.
   a) Choose the initial city at random as the current city.
   b) Determine which of the cities in the edge list of the current city has the fewest entries in its own edge list. The city with the fewest entries becomes the current city. Ties are broken randomly.

Simple Recombination (Crossover)
-*Example*

Reproduce a child using the two parents as follows:





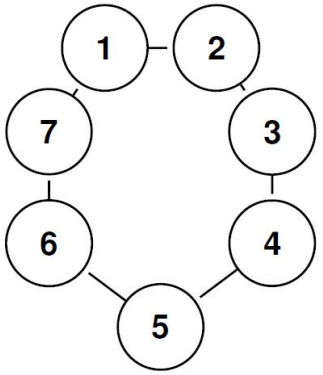
Parent 1	1	2	3	4	5	6	7

$(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$

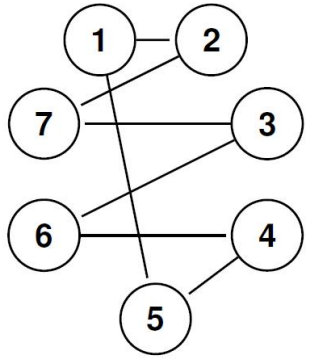
Parent 2	1	2	7	3	6	4	5

$(x_1, x_2, x_7, x_3, x_6, x_4, x_5)$

Example
-Step 1: Construct an Edge Map from 2 Parental Tours



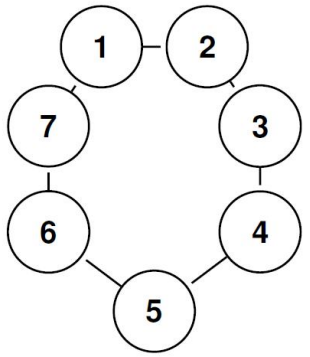
Parent 1: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$



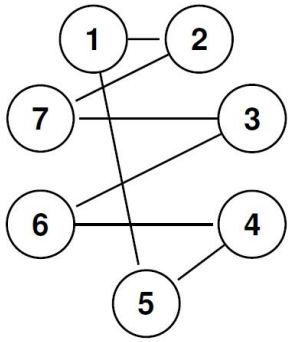
Parent 2: $(x_1, x_2, x_7, x_3, x_6, x_4, x_5)$

Edge Map	
1:	2, 7, 5
2:	1, 3, 7
3:	2, 4, 6, 7
4:	3, 5, 6
5:	4, 6, 1
6:	5, 7, 3, 4
7:	1, 6, 2, 3

Example
-Step a): Choose the Initial City at Random as the Current City



Parent 1: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$



Parent 2: $(x_1, x_2, x_7, x_3, x_6, x_4, x_5)$

	Edge Map
1:	2, 7, 5
2:	1, 3, 7
3:	2, 4, 6, 7
4:	3, 5, 6
5:	4, 6, 1
6:	5, 7, 3, 4
7:	1, 6, 2, 3

- **Select the 1st to visit:**
  - **Candidates:** $1, 2, \ldots, 7$
  - **Decision:** 1
  - **Child:** $(x_1, ?, ?, ?, ?, ?, ?)$

An Example

-Step b): Determine which of the cities in the edge list of the current city has the fewest entries in its own edge list. The city with the fewest entries becomes the current city. Ties are broken randomly.

- **Position/order: 2**
- **Candidates:** $2, 7, 5$
- **Decision:** 2
- **Child:** $(x_1, x_2, ?, ?, ?, ?, ?)$

	Edge Map
1:	2, 7, 5
2:	1, 3, 7
3:	2, 4, 6, 7
4:	3, 5, 6
5:	4, 6, 1
6:	5, 7, 3, 4
7:	1, 6, 2, 3

- **Position/order: 3**
- **Candidates:** ~~1~~, $3, 7$
- **Decision:** 7
- **Child:** $(x_1, x_2, x_7, ?, ?, ?, ?)$

	Edge Map
2:	1, 3, 7
3:	2, 4, 6, 7
4:	3, 5, 6
5:	4, 6, 1
6:	5, 7, 3, 4
7:	1, 6, 2, 3

- **Position/order: 4**
- **Candidates:** ~~1~~, 6,~~2~~,3
- **Decision:** 6
- **Child:** $(x_1, x_2, x_7, x_6, ?, ?, ?)$

	Edge Map
3:	2, 4, 6, 7
4:	3, 5, 6
5:	4, 6, 1
6:	5, 7, 3, 4
7:	1, 6, 2, 3

An Example

-Step b): Determine which of the cities in the edge list of the current city has the fewest entries in its own edge list. The city with the fewest entries becomes the current city. Ties are broken randomly. (continued)

- **Position/Order: 5**
- **Candidates:** $5,\cancel{7},3,4$
- **Decision:** 5
- **Child:** $(x_1, x_2, x_7, x_6, x_5, ?, ?)$

	Edge Map
3:	2, 4, 6, 7
4:	3, 5, 6
5:	4, 6, 1
6:	5, 7, 3, 4

- **Position/Order: 6**
- **Candidates:** $4,\cancel{6,1}$
- **Decision:** 4
- **Child:** $(x_1, x_2, x_7, x_6, x_5, x_4, ?)$

	Edge Map
3:	2, 4, 6, 7
4:	3, 5, 6
5:	4, 6, 1

- **Position/Order: 7**
- **Candidate:** $3,\cancel{5,6}$
- **Decision:** 3
- **Child:** $(x_1, x_2, x_7, x_6, x_5, x_4, x_3)$

	Edge Map
3:	2, 4, 6, 7
4:	3, 5, 6

First Variation of the Recombination

- The ties are broken according to the distance of edges, rather than at random. If ($K \leq n_{x\,i}$) cities have the same number of entries in their edge lists, then city x_j will be chosen according to probability p_{xj}, $1 < j < K$

$$p_{x_j} = \frac{\frac{1}{d_{x_i,x_j}}}{\sum_{k=1}^{K} \frac{1}{d_{x_i,x_k}}} \qquad (1)$$

where

- ✓ x_i is the current city,
- ✓ n_{x_i} is the number of cities in the edge list of x_i and
- ✓ $d_{x_i x_j}$ is the distance between x_i and x_j, $x_i \neq x_j$.

First Variation of the Recombination
-*Example*

Edge Map	
1:	2, 7, 5
2:	1, 3, 7
3:	2, 4, 6, 7
4:	3, 5, 6
5:	4, 6, 1
6:	5, 7, 3, 4
7:	1, 6, 2, 3

	d_{x_1,x_j}	p_{x_j}
j=2	20	$\frac{1/20}{1/20+1/42}$
j=5	42	$\frac{1/42}{1/20+1/42}$

1. Randomly choose a starting city.
   Assume 1 is selected, then child: (x_1, ?, ?, ?, ?, ?, ?)
2. Among {*2, 5*}, choose the next city using the probability distribution {p_{xj}}, j = *2, 5*.
   Assume 2 is selected, then child: (x_1, x_2, ?, ?, ?, ?, ?)
3. ...

Second Variation of the Recombination

- The next city is not chosen according to the number of entries in the edge list, but according to the distances from the current city. That is, the next city x_j will be chosen according to probability p_{xj}, $1 < j < n_{xi}$,

$$p_{x_j} = \frac{\frac{1}{d_{x_i,x_j}}}{\sum_{k=1}^{n_{x_i}} \frac{1}{d_{x_i,x_k}}} \qquad (2)$$

where

- ✓ x_i is the current city,
- ✓ n_{x_i} is the number of cities in the edge list of x_i and
- ✓ $d_{x_i x_j}$ is the distance between x_i and x_j, $x_i \neq x_j$.

Second Variation of the Recombination
-*Example*

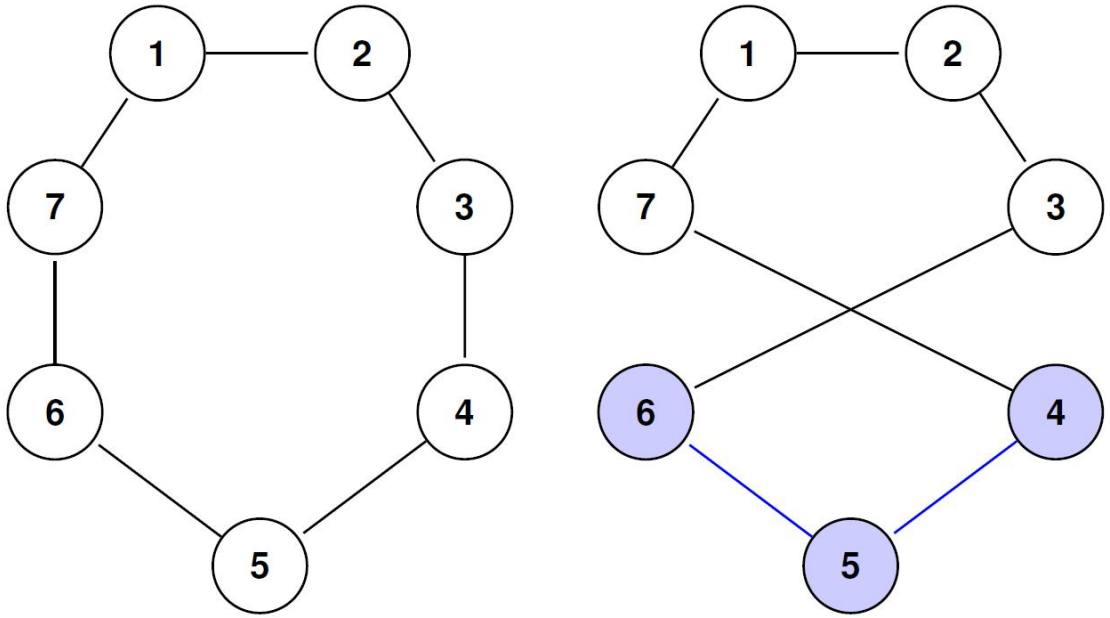
Edge Map	
1:	2, 7, 5
2:	1, 3, 7
3:	2, 4, 6, 7
4:	3, 5, 6
5:	4, 6, 1
6:	5, 7, 3, 4
7:	1, 6, 2, 3

	d_{x_1,x_j}	p_{x_j}
j=2	20	$\frac{1/20}{1/20+1/42+1/35}$
j=5	42	$\frac{1/42}{1/20+1/42+1/35}$
j=7	35	$\frac{1/35}{1/20+1/42+1/35}$

1. Randomly choose a starting city.
   Assume 1 is selected, then child: (x_1, ?, ?, ?, ?, ?, ?)
2. Among {*2, 7, 5*}, choose the next city using the probability distribution {p_{xj}}, *j = 2, 7, 5*.
   Assume 2 is selected, then child: (x_1, x_2, ?, ?, ?, ?, ?)
3. ...

Mutation

Reverse any segment of the tour. Example:



$$(x_1, x_2, x_3, x_4, x_5, x_6, x_7) \rightarrow (x_1, x_2, x_3, x_6, x_5, x_4, x_7)$$

Tournament Selection

- At generation *t*, the population is G_t, of *P* solutions (tours)
  1. Generate *P* solutions by search operator(s).
  2. Each of the *2P* solutions (parents+offspring) competes against *α%* of the *2P* which are chosen at random. Note that:
     - ✓ A solution does not compete against itself.
     - ✓ The probability of s_i obtaining a win over the opponent s_o is

$$p_{s_i} = \frac{tourLength(s_o)}{tourLength(s_i) + tourLength(s_o)}.$$

     - ✓ Solutions with shorter tours have higher probabilities of winning.
     - ✓ *α* is a control parameter.
  3. The population G_{t+1} of next generation consist of the *P* solutions with most wins in the competition.

Roulette Wheel Selection

(For students to fill in.)

Rank-based Selection

1. Tours in a population are first sorted in a non-descending order according to their lengths.
2. Then the *i-th* tour is selected with probability

$$p_i = \frac{P - i + 1}{\sum_{k=1}^{P}(P - k + 1)},$$

where *i = 1, 2, . . . , P*.

Experimental Comparison of Operators
-*Solvers: Six EAs*

- EA1: simple recombination operator but no mutation.
- EA2: favour the nearest neighbour but no mutation.
- EA3: tie is broken in favour of a nearer neighbour but no mutation.
- EA4: reverse mutation + tournament selection ($\alpha = 10$) but no recombination.
- EA5: reverse mutation + roulette wheel selection but no recombination.
- EA6: reverse mutation + rank-based selection but no recombination.

Experimental Setup

- Ten randomly generated TSPs with 100 cities.
- The expected tour length is 100.
- Population size P = 10.

Performance Metrics

- Best (found so far);
- Mean (in a population);
- Entropy (in a population):

$$H = \frac{1}{n} \sum_{i=1}^{n} H_i,$$

where n is the number of cities and

$$H_i = -\frac{1}{\log n} \sum_{i=1}^{n} \left(\frac{n_{i,j}}{2P} \log \frac{n_{i,j}}{2P} \right).$$

P is the population size. $n_{i,j}$ is the number of edges connecting city *i* and city *j* in the population.

→ Higher entropy, more information, higher diversity.

Comparison of Recombination Operators

Table 1: Average results, over 10 TSP instances, of running three EAs for 200 generations.

Metric	Algo.	Mean	Std Dev	Std Err
Best	EA1	309.59	13.535	4.280
	EA2	110.48	4.548	1.438
	EA3	126.07	5.025	1.589
Average	EA1	387.03	12.105	3.828
	EA2	127.26	11.465	3.626
	EA3	126.67	6.260	1.980
Entropy	EA1	0.487	0.015	0.005
	EA2	0.200	0.028	0.009
	EA3	0.151	0.000	0.000

Observation from experiments:

1. Probabilistical greediness is beneficial.
2. More greedy$\not\Leftarrow$ lower population diversity.

Comparison of Selection Mechanisms

Table 2: Average results, over 10 TSP instances, of running EA4, EA5 and EA6 for 200 generations.

Metric	Algo.	Mean	Std Dev	Std Err
Best	EA4	479.81	24.825	7.850
	EA5	565.96	15.303	4.839
	EA6	264.40	5.607	1.773
Average	EA4	539.08	25.351	8.017
	EA5	640.35	21.636	6.842
	EA6	274.97	6.260	1.834
Entropy	EA4	0.466	0.053	0.017
	EA5	0.469	0.030	0.010
	EA6	0.232	0.010	0.003

Observation from experiments:

1. Higher selection pressure is useful, but still outperformed by EA2.
2. Tournament selection is not effective in this case.

Recombination + Mutation

Table 3: Average results of EA3, over 10 TSP instances, with mutation probability 0.002 running for 200 generations.

Metric	Mean	Std Dev	Std Err
Best	121.84	6.006	1.899
Average	126.24	5.605	1.773
Entropy	0.162	0.002	0.001

Observation from experiments:

1. Crossover + mutation work well for TSPs.
2. Probabilistically greedy crossover produces useful "local selection pressure".

More General: Vehicle Routing Problem (VRP) [2]

- 车辆路径问题.
- Aim: find the optimal set of routes for a fleet of vehicles to traverse in order to deliver to a given set of customers/goods.
- NP-hard.

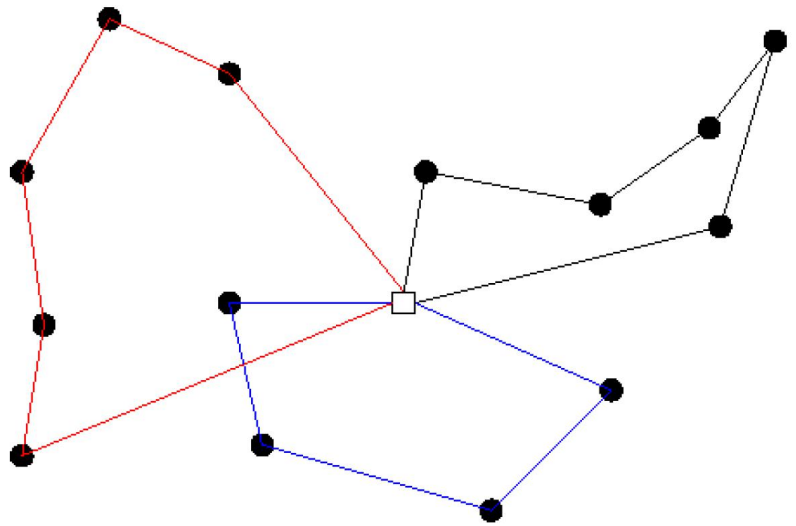


Figure 2: VRP (figure from Wikipedia).

Outline of This Lecture

- Brief Introduction to Combinatorial Optimisation
- Travelling Salesman Problem
- **Cutting Stock Problem**
- Summary

Cutting Stock Problem (CSP)

- Aim: cut an object made of material that can be a reel of wire, paper, or a piece of wood, etc., to satisfy customers' demands.
- Material: the stock of “(large) object”.
- Demands: “(small) items”.

If there is more than one stock length to be cut to fulfil the requests, the problem is called a multiple stock length CSP.

CSP without Contiguity

Minimise the following cost function:

$$\text{Cost} = \frac{1}{m+1}\left(\sum_{j=1}^{m}\sqrt{\frac{w_j}{L_j}} + \sum_{j=1}^{m}\frac{V_j}{m}\right),$$

where

$$\begin{aligned} w_j &= L_j - \sum_{i=1}^{n} x_{ij} l_i, j = 1, \ldots, m \\ V_j &= 1 \text{ if } w_j > 0, \text{ otherwise } 0, j = 1, \ldots, m \end{aligned}$$

subject to $\sum_{j=1}^{m} x_{ij} = N_i, i = 1, \ldots, n$, where n is the number of different requested items, m the total number of stocks cut, w_j the wastage of the j-th stock, V_j the j-th stock with wastage, L_j the stock length of the j-th stock, x_{ij} the number of the i-th requested item in the j-th stock, l_i. the length of the i-th requested item, and N_i. the total number of the i-th requested item.

Contiguity

- In addition to the issue of single vs. multiple stock lengths, contiguity is another important issue in the CSP.
- When a list of items are cut we need to consider storage space requirements where partially finished items are placed until the requested number of items is completed or the packaging size is met. This is referred to as the contiguity issue in CSP.
- It is a more realistic model of real world situations, especially when the problem size becomes large.

CSP

Minimise the following cost function:

$$\text{Cost} = \frac{1}{m+10}\left(\sum_{j=1}^{m}\sqrt{\frac{w_j}{L_j}} + \frac{10}{m}\sum_{j=1}^{m}\left(\frac{o_j}{n}\right)^2\right),$$

where

$$o_j = \sum_{i=1}^{n} y_i$$

$$y_i = 0 \text{ if } \sum_{k=1}^{j} x_{ik} = 0 \text{ or } \sum_{k=1}^{j} x_{ik} = N_i, \text{ otherwise } 1,$$

where o_j is the number of partially finished (open) orders up to the j-th cutting stock, and y_i is the open status of the i-th requested item.

Representation

Example of order-based representation and the mapping from the representation to a solution.

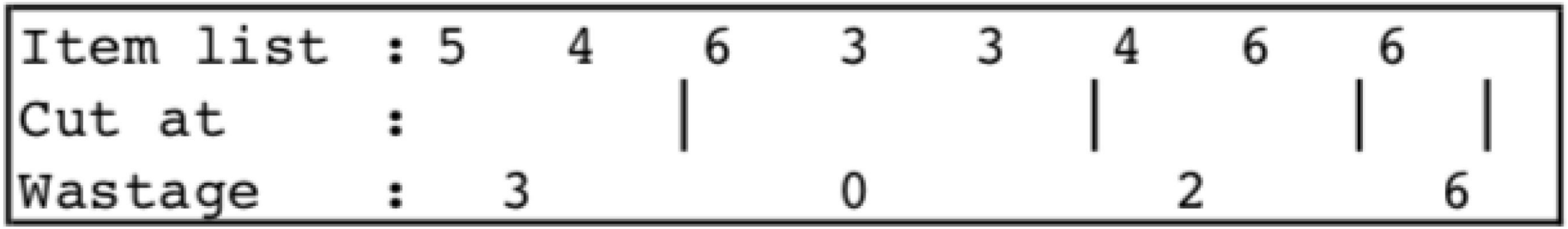


Figure 3: Eight requested items are represented as an ordered list. The stock length is 12. Four stocks are used here. The total wastage is 11.

Swap Mutation

- Idea comes from the concept of distance between two permutation lists.
- Given two permutations with n elements

$$X = (x_1, \cdots, x_i, \cdots, x_j, \cdots, x_n), \text{ and}$$

$$Y = (y_1, \cdots, y_i, \cdots, y_j, \cdots, y_n), 1 < i < j < n,$$

the minimum distance between them can be defined as $D(X, Y) = 1$ if $x_i = y_j$, $x_j = y_i$, and $x_k = y_k$ for all other k's.

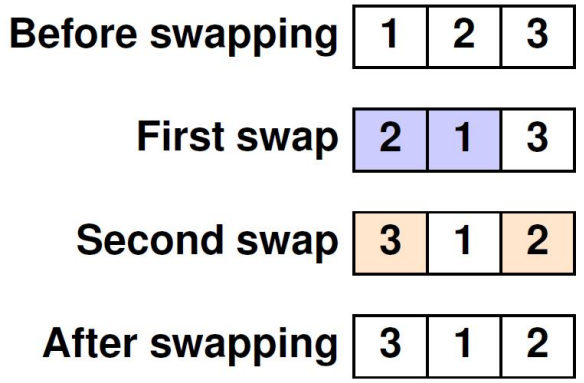
- In other words, the distance between two permutation lists is determined by the minimum number of pair-wise swaps to go from one list to the other.

3 Point Swap (3PS) Mutation

- 3PS swaps the first with the second one, and then swaps the new first one with the third one.
- The first item is selected uniformly at random from an ordered list.
- The second and third items are both selected in two steps:
  1. A stock *j* is selected at random according to the following probability:

$$p_j = \frac{\sqrt{\frac{1}{w_j}}}{\sum_{k=1}^{m} \sqrt{\frac{1}{w_k}}}, \forall w_j \neq 0, \tag{3}$$

  where w_j is the wastage of the *j*-th stock.
  2. Then an item is selected uniformly at random from the stock.



Dealing With Contiguity

- For CSP with contiguity, another mutation operator, the stock remove and insert (SRI) operator, is needed to consider the contiguity.
- To reduce the number of partially finished items while minimising the total wastage, the best way is to appropriately rearrange the cutting sequence. The SRI operator does exactly this. It does not change any stock wastage w_j.
- Steps:
  1. Select an item uniformly at random from an ordered list.
  2. Remove the stock that cuts the selected item.
  3. Search through the list to find the stock that cuts an item of the same length.
  4. Insert the removed stock right behind the first such found stock.

Mutation Size

Similar to the design of 3PS-based mutation, we may have more than one SRI within a single mutation in order to enhance the exploration ability of the SRI mutation.

→ Sounds like an ensemble!

Example: use 4 SRI in each mutation.

Outline of This Lecture

- Brief Introduction to Combinatorial Optimisation
- Travelling Salesman Problem
- Cutting Stock Problem
- **Summary**

Summary

1. Representation is as important as search operators! In fact, it is the combination of good representation, search operators and parameter settings that make an EA work. We should talk about one while keeping others in mind.
2. Combinatorial optimisation problems often required specialised search operators.
3. Principles for designing good search operators are the same as those for designing search operators for continuous problems.

References for This Lecture I

1. Mandell Bellmore and George L Nemhauser. “The traveling salesman problem: a survey”. In: Operations Research 16.3 (1968), pp. 538–558.
2. George B Dantzig and John H Ramser. “The truck dispatching problem”. In: Management science 6.1 (1959), pp. 80–91.

Essential Reading for This Lecture

- X. Yao. “An empirical study of genetic operators in genetic algorithms,” Microprocessing and Microprogramming, 38(1-5):707-714.
- Liang, K. H., Yao, X., Newton, C., & Hoffman, D. (2002). “A new evolutionary approach to cutting stock problems with and without contiguity,” Computers & Operations Research, 29(12), 1641-1659.

Essential Reading for Next Lecture

1. P. Darwen and X. Yao, “A dilemma for fitness sharing with a scaling function,” Proc. of 1995 IEEE Conference on Evolutionary Computation (ICEC’ 95), Dec. 1995, Perth, Australia, IEEE Press, pp.166–171.
2. P. Darwen and X. Yao, “Every niching method has its niche: fitness sharing and implicit sharing compared,” Lecture Notes in Computer Science, Vol.1141, Proc. of Parallel Problem Solving from Nature (PPSN) IV, Springer-Verlag, Berlin, pp.398-407, 1996.