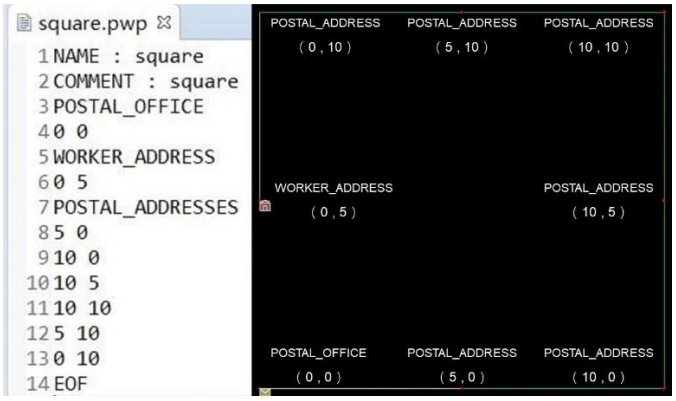
1. (SHORT) QUESTION AND EXAMPLE BASED ESSAY

1.1 RANDOM INITIALISATION:

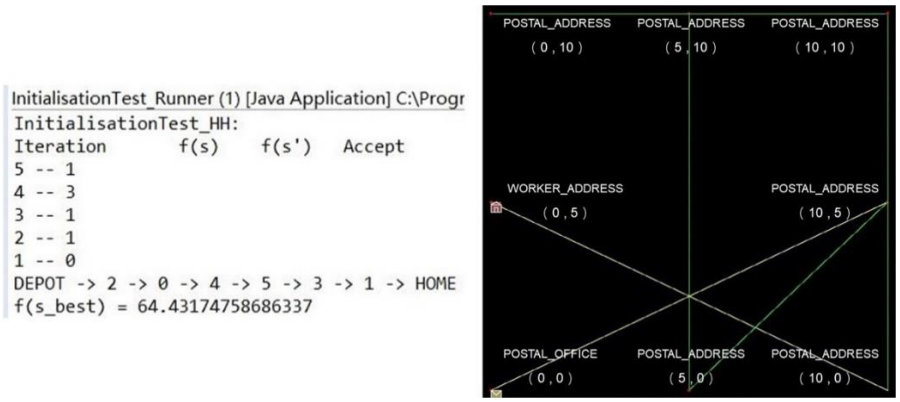
1) In solution $s_1 = [0, 1, 2, 3, 4, 5]$, six POSTAL_ADDRESSES are sequentially mapped onto integers from 0 to 5 in the order of them listed in "square.pwp". For example, for the 3^{rd} read in POSTAL_ADDRESSES, (10, 5), it will be mapped onto location 2, which is $s_1[2]$.
   Postal workers always start their shift at POSTAL_OFFICE and finish their day by returning to WORKER_ADDRESS. Consequently, these two locations are implicitly located before the head and after the tail of s_1.

2) The picture below demonstrates the graphical representation of $s_1 = [\,0, 1, 2, 3, 4, 5\,]$:



3) The implementation of creating a new solution could be explained in two steps. First, create a route containing all location indexes in ascending order. Second, apply the Fisher-Yates shuffle to the previously generated route, which produces a random route by swapping each of the element stores within the route with another randomly selected one before it.
   For example, when a new solution is created $s_0 = [\,0, 1, 2, 3, 4, 5\,]$.
   Swap elements on index 5 and 1, the solution becomes $s_0 = [\,0, 5, 2, 3, 4, 1\,]$.
   Swap elements on index 4 and 3, the solution becomes $s_0 = [\,0, 5, 2, 4, 3, 1\,]$.
   Swap elements on index 3 and 1, the solution becomes $s_0 = [\,0, 4, 2, 5, 3, 1\,]$.
   Swap elements on index 2 and 1, the solution becomes $s_0 = [\,0, 2, 4, 5, 3, 1\,]$.
   Swap elements on index 1 and 0, the solution becomes $s_0 = [\,2, 0, 4, 5, 3, 1\,]$.

4) The picture below demonstrates the graphical representation of $s_0 = [\,2, 0, 4, 5, 3, 1\,]$:



1.2 INVERSION MUTATION

In this project, a loop algorithm implements the inversion mutation. Within this loop, the algorithm maintains a head index and tail index to keep track of the portion of the route that still needs to be revered. In each iteration, this algorithm swaps the integer stored in the head index and tail index then updates these indexes' value. This loop will terminate after the entire target portion of the route has been revered (The head index meets the tail index). An example with solution $s = [\,0, 4, 1, 3, 5, 2\,]$ is demonstrated below.

Step 1: Randomly generate a head index and a tail index to specify the portion of solution representation (Route in the picture) that needs to apply inversion manipulation. In this instance, the head index is set to 1, and the tail index is set to 5.

Step 2: Swap the integers store in the head index and tail index, which is $s[1]$ and $s[5]$. Consequently, the solution becomes $s = [\,0, 2, 1, 3, 5, 4\,]$.

Step 3: Increase the head index by one, then decrease the tail index by one. Therefore, the head index becomes 2, and the tail index becomes 4.

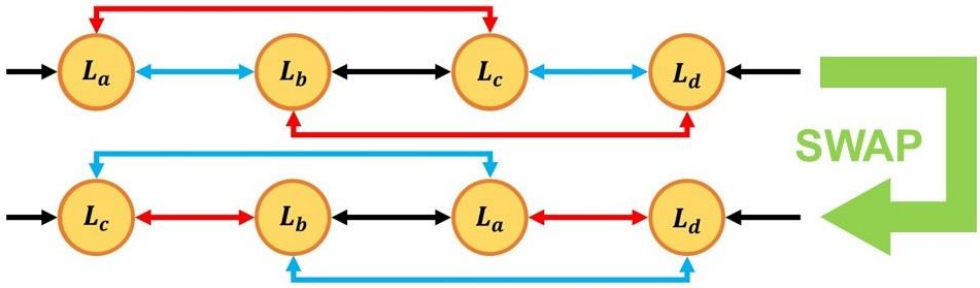
Step 4: Swap the integers store in the head index and tail index, which is $s[2]$ and $s[4]$. Consequently, the solution becomes $s = [\,0, 2, 5, 3, 1, 4\,]$.

Step 5: Increase the head index by one, then decrease the tail index by one again. Therefore, the head index becomes 3, and the tail index becomes 3 as well.

Step 6: The head index is no longer smaller than the tail index. Therefore, the loop algorithm terminates, and the output solution after inversion mutation becomes $s = [\,0, 2, 5, 3, 1, 4\,]$.

1.3 DELTA EVALUATION FOR ADJACENT SWAP

The essence of delta evaluation is to improve calculation efficiency by preserving previous efforts. An example is demonstrated below.

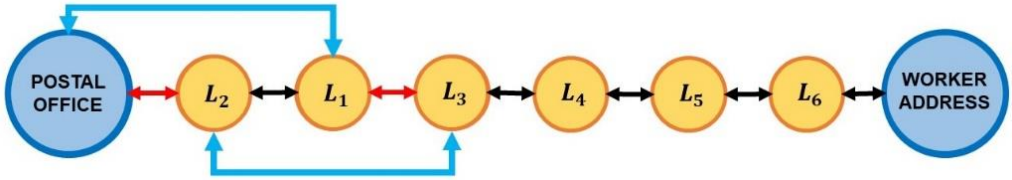


After swap location L_c with location L_b, some route segments have been preserved (Black lines), two route segments are disappeared (Blue lines), and two route segments are

newly appeared (Red lines). Consequently, the objective value could be updated efficiently by adding new route segments' costs to the original value and subtracting the disappeared ones' costs immediately.

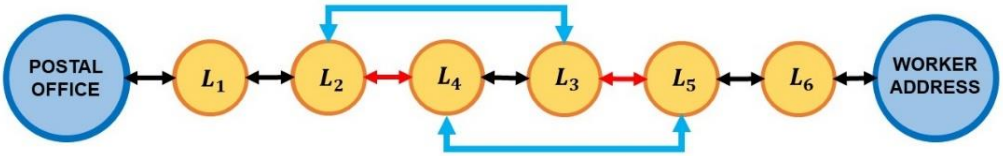
Here are the explanations of delta evaluation operations in three specific scenarios of a problem instance with six delivery locations $(L_1, L_2, L_3, L_4, L_5, L_6)$, the disappeared segments are blue, the newly appeared ones are red. PO and WA represent the POSTAL_OFFICE and WORKER_ADDRESS locations.

1) Adjacent swap of locations L_1, L_2 :



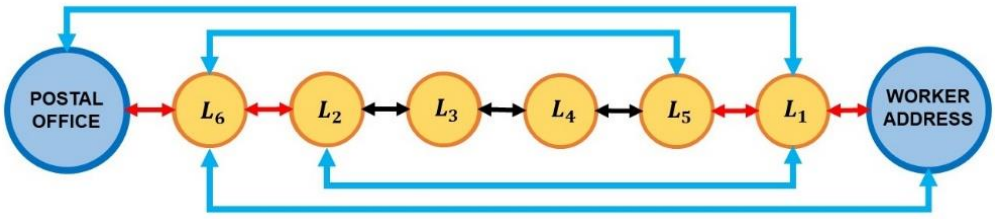
$$f(s_{i+1}) = f(s_i) + C(PO, L_2) + C(PO, L_2) - C(PO, L_1) - C(L_2, L_3)$$

2) Adjacent swap of locations L_3, L_4 :



$$f(s_{i+1}) = f(s_i) + C(L_2, L_4) + C(L_3, L_5) - C(L_2, L_3) - C(L_4, L_5)$$

3) Adjacent swap of locations L_6, L_1 :



This instance is different from previous ones because it has swapped two non-adjacent locations. Within this instance, four route segments are disappeared (Blue lines), and four segments are newly appeared (Red lines).

$$f(s_{i+1}) = f(s_i) + C(PO, L_6) + C(L_6, L_2) + C(L_5, L_1) + C(L_1, WA) - C(PO, L_1) - C(L_5, L_6) - C(L_1, L_2) - C(L_6, WA)$$

2. TRAMSTOPS-85

(1) A single-point based iterative local search hyper-heuristic structure has been applied as the design of this project.

During the beginning of designing, both iterative local search and evolutionary algorithm were considered to be applied as the design framework. According to

sufficient online research, both of them perform well in solving TSP. However, according to the lecture slides in lecture 5 and some observation with crossover operators, genetic algorithm with crossover performs poorly in solving PWP. This is possibly caused by the randomness of the crossover operators which impairs the performance of evolutionary algorithms. Moreover, inspired by the SR_IE hyper-heuristic, the performance of hyper-heuristic could be improved by increasing the number of search iterations. Consequently, the iterative local search is selected to the design framework.

Besides choosing the iterative local search structure to be the hyper-heuristic framework, a perturbating component has been designed to control the value of the intensity of mutation. This design is inspired by the observation of the process of solving PWP. All solutions should be randomly initialized at the beginning of searching, and the population could hardly obtain the optimal solution. Under this situation, increase the intensity of mutation should be highly possible to discover better routes. At the end of searching, all solutions should be close to one of the better routes or at least partly following a better sequence. Since the mutation heuristic operators are purely random, intensive mutations could hardly generate better routes under this kind of circumstance. Therefore, mutation intensity should be gradually decreased at the end of the searching process.

By observing different problem instances, the effect of mutation with the same intensity decreases as the number of locations with the problem instance increases. So, an increment unit is developed to control the size of mutation intensity, which guaranteed the effectiveness of mutation operators.

(2)

a. Roulette wheel selection has been chosen to be the heuristic selection method of this hyper-heuristic.

   Greedy selection, reinforcement learning selection and choice function selection method have been considered as the candidates of the heuristic selection method. Because greedy selection does not involve a learning process, it could hardly improve its efficiency throughout the execution. As a result, greedy selection is not considered. Since choice function must maintain each heuristic's individual performance, cooperative performance and elapsed time, it needs more storage space and calculation power throughout the searching process. Choice function selection is not selected to improve the operation speed of this hyper-heuristic. Reinforcement learning selection is preferred since it utilizes a robust evaluation mechanism with high execution speed. Furthermore, an example has been implemented in the previous lab session (lab07), which could be applied as a reference for realization. Roulette wheel selection is chosen to be the specific reinforcement learning selection method. Some changes have been made to its

score update system, which enables it to cooperate will simulate annealing. The score of heuristic pairs will only be increased when the best solution is updated.

b. Simulated annealing with Lundy & Mees cooling schedule has been applied as the move acceptance method of this hyper-heuristic.

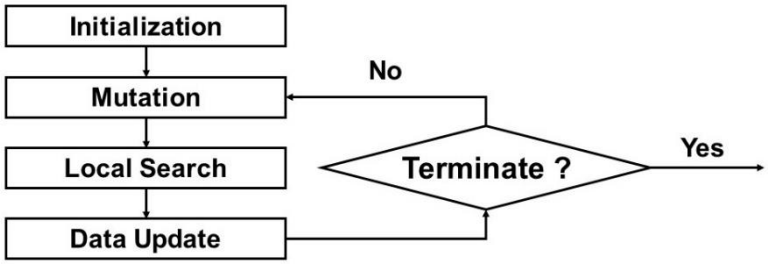
   Comparing to non-stochastic basic and non-stochastic threshold ones, stochastic move acceptance methods should perform better with PWP. Because it involves random elements, and its performance could be significantly enhanced by improving the parameters within. A dynamic stochastic move acceptance is selected to ensure this method could perform analogously with different problem instances. Adaptive ones are not considered because the execution time is limit to one minute. An adaptive process such as reheating could waste execution time in updating the number of non-improve actions, and it could hardly guarantee that a better result will be produced. The choice of cooling schedule is generated from previous lab sessions. According to the experiments conducted in lab 03, Lundy & Mees cooling should perform better than linear cooling and geometric cooling.

c. A meme structure with six different low-level heuristic pairs is applied in this hyper-heuristic. The combination of low-level heuristic operators within each heuristic pair is listed below:

   1. Adjacent swap with next descent local search
   2. Adjacent swap with Davis's hill climbing
   3. Inversion mutation with next descent local search
   4. Inversion mutation with Davis's hill climbing
   5. Random reinsertion with next descent local search
   6. Random reinsertion with Davis's hill climbing

   According to exercises from previous lab sessions (lab05) and lecture slides (lecture 06), hyper-heuristic algorithms with meme has several advantages over other algorithms. First, by applying this structure, the cooperative performance between different low-level heuristic operators could be scored and maintained by the roulette wheel heuristic selection method. This enhanced the performance of the learning mechanism of this hyper-heuristic. Second, there exist only six combinations of low-level heuristic operators, which guarantees that all heuristic pairs could be thoroughly evaluated with limited execution time. Therefore, three mutation heuristic operators and two local search operators are combined with each other to generate six heuristic pairs. And the heuristic selection method will score each of these heuristic pairs according to their performance during the execution.

(3) The picture below demonstrates the frame structure of this hyper-heuristic.



Initialization:

This is the first stage of the hyper-heuristic, which initialize the solution and update its object value. The heuristic selection and move acceptance method will be created, and a heuristic pair will be generated simultaneously.

Mutation:

A mutation heuristic operator indicates by the previously generated heuristic pair will be applied to current solution. The result will be stored in the candidate solution index for later local search process.

Local Search:

A local search heuristic operator indicates by the heuristic pair will be applied to the solution stored in the candidate position. After that, the move acceptance method will decide whether to accept it or not. If the candidate solution is accepted, it will be copied back to the original solution's index.

Date Update:

If the best solution is updated, the score of current heuristic pair and the intensive of mutation will be increased. Otherwise, both of them will be decreased. New heuristic pair will also be generated after the scoring, and the acceptance method will be advanced in this stage as well.

Terminate:

This program will terminate if the time has expired. Otherwise, it will jump back to the mutation stage.

Words count: 1738