AIM coursework report

1. Pseudo-Code

1.1 Pseudo-code of MemeticAlgorithm():

```
Input: current problem p1        /* the problem that should be solved */

Set population[POP_SIZE];        /* the current population */
Set offspring[POP_SIZE];         /* the offspring of the population */
Set gen = 0;

generate(population(0));     /* initial population */

While gen < MAX_NUM_OF_GEN && time_spent < MAX_TIME Do
    gen++;
    tournament();  /* select to be the original offspring(mating pool) */
    crossover();     /* offspring[] are replaced by the productions */
    mutate();
    feasibility_repair();     /* repair the solutions in offspring[] */
    variable_neighbourhood_search();       /* do the VNS */
    replace();              /* generate the next population: population(gen + 1) */
    find_best_sln();          /* find the best solution in the offspring */
EndWhile

Output: Best_sln
```

1.2 Pseudo-code of some functions in MA

Pseudo-code of **generate( ):**

```
Input: current problem p1

/* the initial population is generated by two parts: greedy_heuristic and its neighbor;
randomly generated. And the size of each part is determined by the population size
and the rand rate. */

Set pop_rand_size = POP_SIZE * POP_RAND_RATE;
Set pop_neighb_size = POP_SIZE – pop_rand_size;
Set count = 0;
curt_solution = greedy_heuristic();
```

```
/* find neighbors by 1 − 1 swap */
For i = 0 to prob−> n Do
    If count >= pop_neighb_size
        Then break; /* count equals to pop_neighb_size, stop finding neighbors */
    EndIf
    For j = 0 to prob−> n Do      /* 1 − 1 swap */
        /* swap i & j and find the first feasible neighbors of the curt_solution */
    EndFor
    count + +;
EndFor

/* randomly generation */
While count < POP_SIZE Do
    /* randomly generate the solution when the left capacity is enough */
EndWhile
```

Pseudo-code of **crossover( ):**

```
For i = 0 to POP_SIZE Do
    rand = rand_01();
    If rand < CROSSOVER_RATE && has_two_parents()
        Then ux();      /* do the uniform crossover */
        Else            /* keep the current solution */
    EndIf
EndFor
```

Pseudo-code of **ux( ):**

```
Input: parent solutions s1, s2

/* uniform crossover */
For i = 0 to chorom_len
    rand = rand_01();
    If rand >= 0.5
        Then exchange(s1−> x[i], s2−> x[i]);
    EndIf
EndFor
```

Pseudo-code of **feasibility_repair( ):**

```
/* repair the solution by removing the least favorable items */
Input: current problem p1

For i = 0 to p1−> n Do          /* calculate the ratio each item */
```

```
    ratio = value/size;
EndFor
qsort();                          /* sort the item by ratio, in increasing order */

ratio_index[p1-> n];        /* store item index when the items sorted by ratio */
For i = 0 to p1-> n Do
    ratio_index[i] = p1-> items[i].indx;
EndFor
qsort()                           /* resort the item original order by sorting by index */

For k = 0 to POP_SIZE Do

  If offspring[k]-> feasibility != 1   /* check every solution in offspring[] */
    Then change_chrom(p1-> x[ratio_index[k]]);     /* remove the least favorable items */
         Evaluate_solution( );
         If offspring[k]-> feasibility != 1
            Then continue; /* if feasibility still not 1, continue repairing */
            Else break;
         EndIf
  EndIf
EndFor
```

Pseudo-code of **variable_neighbourhood_search( ):**

```
Set nb_indx = 0;
While nb_indx < 3 Do        /* nb_1: insert new items; nb_2: 1 - 1 swap; nb_3: 1 - 2 swap & 2 - 1 swap */
    best_descent_vns(nb_indx + 1);
    If neighb_sln better than curt_sln
        Then copy_solution;
             nb_indx = 1;
        Else nb_indx++;
    EndIf
EndWhile
```

2. Description of MA

2.1 For GA

Learned from lectures and some literatures, the implementation of GA is encoding, generate the initial population, and do the operations like selection, crossover, mutation and replacement to the current population and then generate the next population.

Generally, the initial population should generate randomly, but it is found that the initial population can affect the result a lot. So, in my GA, in order to optimize the algorithm, I set a POP_RAND_RATE, divide the generate to two parts. One part generates randomly, the other part generates from greedy heuristic and its neighbors, which can have a better and diversity initial population.

For selection, I choose to use tournament selection. Each time choose 3 individuals randomly and select the best of them into the mating pool.

For crossover, the uniform crossover is used. When do the uniform crossover, each bit of the parent chromosome for exchange with a probability of 0.5.

After crossover and mutation, some individuals become infeasible, and the function *feasibility_repair( )* is used to remove the least favorable items to make it feasible.

When finishing above operations, function *replace( )* combines the current population and its offspring, then select the best POP_SIZE individuals to be the next population.

2.2 For local search

In order to get better accuracy, VNS is used for the local search in my MA. It uses insert new items, 1-1 swap and 1-2 swap & 2-1 swap to find the neighbor of current solution. When inserting new items cannot get better solution, then start to do 1-1 swap. After 1-1 swap, if there is no better solution, the algorithm will do 1-2 swap & 2-1swap.

And in both 1-1, 1-2, 2-1 swap and insert new items, best descent is used.

3. Parameter Tuning Process

3.1 POP_SIZE

Test instance: mknapcb8-problem[6]

Test	POP_SIZE	CROSSOVER_RATE	MUTATION_RATE	Gap	Generation
1	10	0.9	0.01	1.1689%	431
2	30	0.9	0.01	1.0215%	88
3	50	0.9	0.01	0.6022%	66
4	60	0.9	0.01	1.0499%	40

When the POP_SIZE is small, i.e. 10, the speed of MA will be faster, but diversity of the population will decrease and make the accuracy reduce. But if with too large POP_SIZE, the iterations will reduce and make the optimal solution harder to find. After some experiments, the POP_SIZE 50 is most suitable for my MA.

3.2 CROSSOVER_RATE (when POP_SIZE is 50)

Test instances: mknapcb8-problem[6]

Test	POP_SIZE	CROSSOVER_RATE	MUTATION_RATE	Gap
1	50	0.4	0.01	0.7159%
2	50	0.6	0.01	0.7354%
3	50	0.8	0.01	1.1290%
4	50	0.9	0.01	0.6022%

With the POP_SIZE is 50, the results of changing CROSSOVER_RATE are shown above. Larger CROSSOVER can make the offspring more diversiform. And in my MA, the CROSSOVER_RATE 0.9 is the most suitable and can reach better solution.

3.3 MUTATION_RATE (when POP_SIZE is 50 and CROSSOVER_RATE is 0.9)

Test instances: mknapcb8-problem[6]

Test	POP_SIZE	CROSSOVER_RATE	MUTATION_RATE	Gap
1	50	0.9	0.005	1.1298%
2	50	0.9	0.01	0.6022%
3	50	0.9	0.03	1.01%
4	50	0.9	0.05	1.24%

When setting POP_SIZE to 50, CROSSOVER_RATE to 0.9, test the MUTATION_RATE. If the MUTATION_RATE is large, some solutions have more probability to be changed and sometimes the result will get worse after lots of mutation. But if the MUTATION_RATE is too small(0.005), it can be regarded as no mutation, and it will reduce the diversity of the offspring. As shown in figure, the most suitable MUTATION_RATE is 0.01 in my MA.

4. Result

The results of my MA for each file are as follows shown by table.

file	max_gap	min_gap	average	run_time_of_each_problem	experimental environment
mknapcb1	0.2827%	0.0000%	0.0542%	30s	macOS
mknapcb2	0.3931%	-0.0000003%	0.0884%	120s	macOS
mknapcb3	0.3232%	0.0409%	0.1276%	300s	macOS
mknapcb4	1.0756%	0.0000%	0.2462%	60s	macOS
mknapcb5	0.5629%	0.0282%	0.2037%	180s	macOS
mknapcb6	0.5589%	0.0926%	0.3481%	300s	cslinux
mknapcb7	1.3861%	0.0000%	0.5450%	60s	macOS
mknapcb8	1.2079%	0.1166%	0.3593%	300s	macOS

And I want to point out that, when running mknapcb2, problem[21] on my own computer with macOS, the optimal solution that the MA found (155944) is better than the given best solution(155940). And I used the mk_checker, the solution is feasible!

5. Reflection of GA/MA

5.1 Strengths

a. In GA/MA, the operation objects(solutions) are encoded in linear chromosomes of fixed length, which makes the algorithms have wide range of application, such as processing image and machine learning.
b. The search starts from the initial population instead of one individual, which has a relatively large coverage of solutions and is beneficial to find the global optimal solution.
c. In every generation of GA/MA, the algorithm operates multiple individuals in the population, that reduces the risk of falling into the local optimal solution and makes it achieve parallelization.
d. Use the fitness function to evaluate an individual which is not constrained by some calculate of the derivative and easy to implement.
e. GA/MA based on probability but not certainty rule, so the search is random, and also can reduce the risk of falling into the local optimal solution.
f. GA/MA is extendible, because many parts of these algorithms can combine with other algorithms to make the algorithm better. i.e. When generating the initial population, greedy search can be used. And when doing local search, many different ways to find neighbor or do the descent can be used.

5.2 Weaknesses

a. The implementation of encoding of some problems is a little complex.
b. Parameters have a deep effect on the result, such as CROSSOVER_RATE, POP_SIZE, MUTATE_RATE. And it will take kind a long time to find the most suitable parameters.
c. When doing complex local search, the speed gets slowly and should take more time to find the optimal solution.
d. The results are partly dependent on the initial population.
e. It may premature convergence when GA/MA.