**Report**

Tianlang Tan

*20028268*

*scytt1@nottingham.edu.cn*

1. **Description**

This coursework is to solve Multi-dimensional knapsack problem by implementing memetic algorithm (MA) which is a variant of genetic algorithm. Memetic algorithm is combined with genetic algorithm (GA) and local search. In genetic algorithm, the main procedure is divided into encoding, initialization, selection, crossover, mutation and replacement. This basic idea of GA is to simulate natural selection that choose the better individual to survive. As for local search, it can find the local optimal in each generation. In this coursework, solution for Multi-dimensional knapsack problem is represented by binary representation. Initialization for all the solution are random. Tournament selection is selected as the selection operation. Uniform crossover is selected as the crossover operation. Mutation is operated randomly. Replacement is implemented to replace the best individuals to next generation. As for local search, this course uses variable neighborhood search as the local search method.

1. **Pseudo code for memetic algorithm:**

n: number of instances in the problem file

load the instances to Ins(i) (i = 0, 1, 2 ... n)

for each instance Ins(i)

memeticAlgorithm(Ins(i)){

    init\_population(Ins(i), parent\_pop); // initialize instance to a population

    while(iteration < itermax && time\_spent < MAX\_TIME){

        selection(mating\_pool, parent\_pop);  //tournament selection

        crossover(mating\_pool);  //uniform crossover

        mutation(mating\_pool);

        feasibility\_repair(mating\_pool); //keep droping the item with minimal price until the objective is within the constrain in all dimensions

        varaible\_neighbourhood\_search(mating\_pool);    //use pair swap and 1-2 swap

        replacement(mating\_pool, parent\_pop);   //replace top 50 best to population

    }

    update\_best\_solution(parent\_pop);   //update best solution

}

output\_solution(best\_sln, out\_file);    //output the best solution

free\_operaton();                        //free the memory

init\_population(Ins(i), parent\_pop){

    for each individual in parent\_pop{

        //initialize all item as unpacked

        //randomly pack item into knapsack until the it violate the capacity

    }

}

selection(mating\_pool, parent\_pop){

    for each individual in mating\_pool{

        //initialize this individual

        //select several candidates from parent\_pop

        //packed the best candidate into mating\_pool

    }

}

crossover(mating\_pool){

    //use uniform crossover

    //crossover between the head and tale individual

}

mutation(mating\_pool){

    for each individual in mating\_pool{

        //mutate each chromosome with MUTATION\_RATE

    }

}

feasibility\_repair(mating\_pool){

    for each individual in mating\_pool{

        while(individual is not feasible){

            //find the packed item with minimal price

            //remove the item

            //evalutate the individual whether it violates the constrain

        }

    }

}

varaible\_neighbourhood\_search(mating\_pool){

    for each individual in mating\_pool{

        initialize current\_solution

        while(neighbourhood < K){       //K: index of neibourhood

            neibourhood\_solution = best\_descent\_vns(neibourhood\_index, current\_solution)

            if(neibourhood\_solution > current\_solution){       //jump back to first neibourhood

                current\_solution = neibourhood\_index;

                neibourhood\_index = 1;

            }

            else{       //go to next neibourhood

                neibourhood\_index++;

            }

        }

        individual = current\_solution

    }

}

best\_descent\_vns(neibourhood\_index, current\_solution){

    if(neibourhood\_index == 1){     //pair swap

        //divide the packed and unpacked items into two lists storing it index for reducing the time complexity

        //travas two list to select two items to swap

        //if the swap is better then record this move

        //if the better swap is greater than the best swap, apply this swap

    }

    else if(neibourhood\_index == 2){    //1-2 swap

        //divide the packed and unpacked items into two lists storing it index for reducing the time complexity

        //apply 1-2 swap for 10,000 times to reduce the running time in one iteration

        //if the swap is better then record this move

        //if the better swap is greater than the best swap, apply this swap

    }

}

replacement(mating\_pool, parent\_pop){

    //joint mating\_pool and parent\_pop togeter

    //quick sort the mixed pool

    //select the top 50 solution as the parent\_pop for next generation

}

1. **The result of the algorithm:**

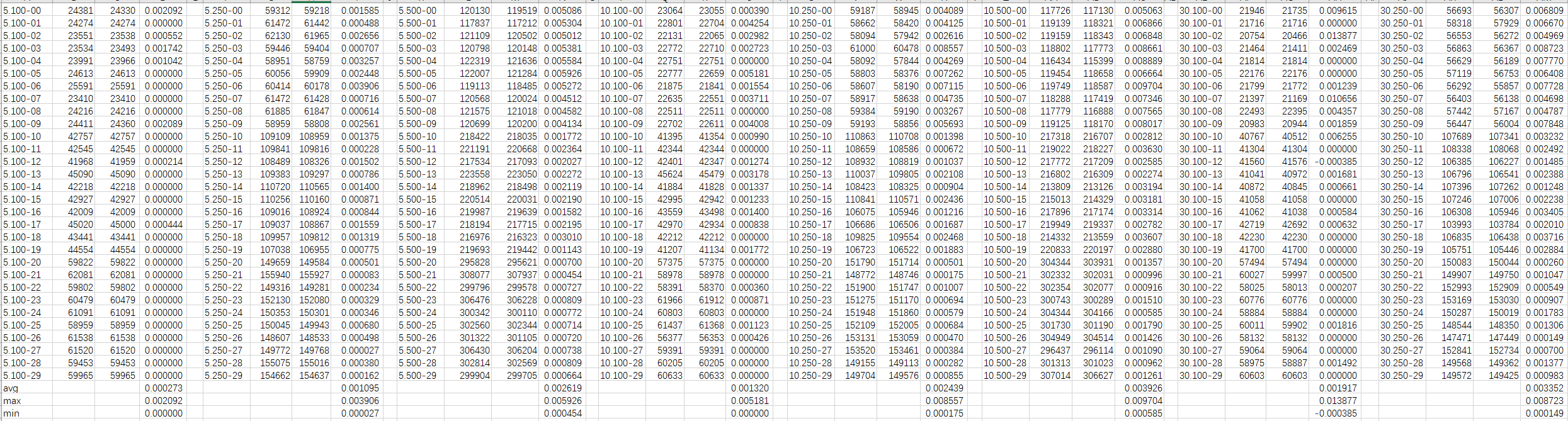
Overall result(x2go):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| file No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Avg gap% | 0.0273 | 0.1095 | 0.2619 | 0.1320 | 0.2439 | 0.3926 | 0.1917 | 0.3352 |
| Max gap% | 0.2092 | 0.3906 | 0.5926 | 0.5181 | 0.8557 | 0.9704 | 1.3877 | 0.8723 |
| Min gap% | 0.0000 | 0.0027 | 0.0454 | 0.0000 | 0.0175 | 0.0585 | -0.0385 | 0.0149 |

Two instances are above 1% which is mknapcb7-02 and mknapcb7-07

One instance produces a better solution which is mknapcb7-12

Detail result(x2go):



1. **tuning process:**

First version:

MA was implemented with tournament selection, one-point crossover, feasibility repair that drop the cheapest items and first descent local search (only have pair swap). In this version, the result is quite bad that the average gap for mknapcb1 is about 5%. Although all the main parameters (CROSSOVER\_RATE, MUTATION\_RATE, POP\_SIZE, TOURM\_SIZE) were tuned, the result is still bad. The main reason is that the pair swap will end in local optimal with fixed number of items which cannot diversify the search for other optimal with different number of items. As a result, VNS is considered due to its variable neighborhood

Second version:

MA was implemented with tournament selection, uniform crossover, feasibility repair that drop the cheapest items then add back items and best\_descent VNS (with 3 neighborhoods 1-swap, 2- swap, 3-swap). In this version, some idea from the paper in the coursework specification (“A genetic algorithm for the multidimensional knapsack problem”) is implemented. Firstly, I use uniform crossover and add operation because it is implemented in the paper. Secondly, an additional add operation is implemented because it can improve the objective value after the drop operation in selection. Thirdly, VNS is implementing without shaking operation. In this case, this running time is too long for one iteration due to the best\_descent VNS have too many for loops and the VNS will keep running until no more improvement is found. Initially, within one population, some of the solutions were randomly selected (about 10%) to perform VNS. After that, the iteration number for one instance increased (from less than 50 iteration to about 100 iteration) whereas the performance is not stable that some of the instance cannot go inside 1%. The reason for this is that for best\_descent VNS cause a lot of time.

Final version:

MA was implemented with tournament selection, uniform crossover, feasibility repair that only drop the cheapest items and best\_descent VNS (with 2 neighborhoods: 2- swap and 1-2-swap). In this version, add operation 1-swap and 2-1swap are removed because they cause too much time and will decrease the diversity of the population. Also, within one solution, a fixed number of 1-2swap is implemented that it will not go through all the feasible 1-2swap for one solution. In other words, this save a lot of time. More iteration can bring better performance.

Finally, the main parameter tuning will be discussed.

CROSSOVER\_RATE: this parameter is often set to 0.5 in uniform crossover.

MUTATION\_RATE: this is the way to diversity the population which is convergent. In this algorithm, it is set to 0.01 because a greater MUTATION\_RATE will destroy the result got from VNS. When MUTATION\_RATE is set to 0.05/0.1, the best solution is worse.

POP\_SIZE: this is the main parameter effecting the running time for each iteration of one instance. In this algorithm, it is set to 50 because this brings the best performance. When POP\_SIZE is set to 100/200, the time for on generation is too long so that the convergency speed is slow

TOURM\_SIZE: this is the parameter effecting the convergent speed. If the TOURM\_SIZE is too large, selection operation has higher rate to select items with large objective. As a result, to diversify the population, TOURM\_SIZE is set to 2. When the TOURM\_SIZE is set to 5/10, the convergent speed is too fast that it will be stuck into local optimal.

VNS\_SWAP\_NUM: this is the parameter decide how many swap will be considered in best\_descent\_vns(). In this algorithm, it is set to 10000 to exploit more swaps. When VNS\_SWAP\_NUM is set to 5000, the best solution is worse.

1. **Pros and Cons of GA/MA methods**
   1. Pros for GA
2. The optimization process is applied on a set of individuals which means it have less chance to stuck into local optimal.
3. Have good performance on global search because of the crossover and selection operation. These two operations will retain good individuals
   1. Cons for GA
4. Have bad performance on local search that the search efficiency is low in late generation.
5. Have many parameters to tune which is based on experience.
   1. Pros for MA
6. Include both global and local search which will have better performance.
   1. Cons for MA
7. Have many parameters to tune which is based on experience.