Labs 4-6: Heuristics & Single Point Metaheuristics

1. Introduction

In the following 3 sessions, you shall practice in using single point metaheuritics to solve a multi-dimensional knapsack problem, which is an extended version of the 1D knapsack problem that you have practiced in labs 2 and 3 and has proved to be NP-Hard. The problem has found applications in cloud computing, networking and other practical scenarios. Like in the past, you are asked to write a C/C++ program to solve this problem. A sketch program (downloadable from Moodle) is provided for you also to help you focus more on algorithm implementation.

2. Multi-dimensional Knapsack Problem

Multi-dimensional knapsack problem is an extension of the 1D knapsack problem by adding capacity constraints in multiple dimensions. The problem can be formally defined as follows. Given a set of n items numbered from 1 up to n, each with a size vector $\mathbf{v}=(v_{1j}, v_{2j}, v_{3j}, \ldots, v_{mj})$ where v_{ij} is the i-th dimensional size of item j. b_i is the i-th dimensional size of knapsack. p_j is the profit of item j if it is included in the knapsack. Denote x_j be the binary variables to indicate whether item j is included in the knapsack (=1) or not (=0). The problem to be solved is then formulated as follows

$$\sum_{j=1}^{n} p_j x_j$$

Subject to:

$$\sum_{j=1}^{n} v_{ij} x_j \leq b_i \qquad i = 1, \ldots, m$$

$$x_j = \{0,1\}$$

3. Problem instances

In this lab, you are asked to attempt some more challenging instances from paper: P.C. Chu and J.E.Beasley "A genetic algorithm for the multidimensional knapsack problem", Journal of Heuristics, vol. 4, 1998, pp63-86. All the data files are compressed in `mknap-`

`instances.zip`, downloadable from Moodle. The zip file includes 9 problem instance files (each containing 30 instances), 1 data format file `file-format.txt` and 1best known solution file `best-feasible-slns.txt`, which can be used as a reference to check the performance of your algorithm.

4. Task 1 (lab 4)

Implement a basic simulated annealing algorithm for this problem. Like in lab 3, the initial solution can be generated by some greedy heuristics. You are suggested to use Lundy-Mees nonlinear cooling function ($t = t/(1 + \beta t)$) where β is an input parameter to decide how fast you cool the temperature. In addition, you need to set the initial temperature t_s and t_f as well as the number of iterations at each temperature, *iter*. Some parameter turning work shall be required if you want to obtain high quality solutions.

5. Task 2 (lab 5)

Implement a tabu search method for this problem. Set the computational time same as the run time of your simulated annealing in Task 2 and compare and contract their results and performance.

7. Task 3 (lab 6)

Implement a Variable Neighbourhood Search (VNS) method with at least 3 different neighbourhoods. Compare and contrast the results and performance with Simulated Annealing and Tabu Search.

8. Experiments conditions and requirements

Since your coursework shall also requires you solve the same problem, it is strongly advised that you follow the following requirements:

(1) Your source code should be properly commented.

(2) Name your program file in the format of "yourid-labx.c" where x is the lab session index. For example, if your student number is 2019560 and it is the solution file for lab 4, name your program as 2019560-lab4.c (or 2019560-lab4.cpp).

(3) Your program should compile without errors on **CSLinx** Server. Therefore, please fully test it before submission. You may use one of the following commands (assuming your student id is 2019560 and your program is named after your id):

```
gcc -std=c99 2019560-lab4.c -o 2019560-lab4
```

or

```
g++ -std=c++11 2019560-lab4.cpp -o 2019560-lab4
```

(4) After compilation, your program should be executable using the following command:

```
./2019560-lab4 -s data_file -o solution_file -t max_time
```

where `2019560-lab4` is the executable file of your program, `data_file` is one of problem instance files specified in Section 3. `max_time` is the maximum time permitted for a single run of your algorithm. `soluton_file` is the file for output the best solutions by your algorithm. The format should be as follows:

```
# of problems
objective value of instance 1
x1 x2 x3 … x_n
objective value of instance 2
x1 x2 x3 … x_n
… …
objective value of last instance
x1 x2 x3 … x_n
```

An example solution file for problem data file "**mknapcb1.txt**" is available on moodle.

(5) The solution file that your algorithm (`solution_file`) is expected to pass a solution checking test successfully using the following command:

```
./mk_checker -s problem_file -c solution_file
```

where problem_file is one of problem data files in Section 3. If your solution file format is correct, you should get the following command line message "**All solutions are feasible with correct objective values.**"

The solution checker can be downloaded from moodle page. The checker is runnable on CSLinux server only.

(6) Your program should take no more than 5 min for a single run (i.e. 5 min max_time is set as your stopping criteria of your algorithm).

8. How to submit

To be confirmed later.