Coursework: Perturbative hyper-heuristic for Bin Packing Problem

1. Introduction

Bin packing is one of the most studied combinatorial optimisation problems and has applications in logistics, space planning, production, cloud computing, etc. Bin packing is proven to be NP-Hard and the actual difficulties depend on both the size of the problem (i.e. the total number of items to be packed) and other factors like the distribution of item sizes in relation to the bin size as well as the number of distinct item sizes (different items may have a same size).

In this coursework, you are asked to write a C/C++/Python program to solve this problem using a **perturbative hyper-heuristic** method. In addition to submitting source code, a written report (no more than 2000 words and 6 pages) is required to describe your algorithm (see Section 4 for detailed requirements). Both your program and report must be completed independently by yourself. The submitted documents must successfully pass a **plagiarism checker** before they can be marked. Once a plagiarism case is established, the academic misconduct policies shall be applied strictly.

This coursework carries 45% of the module marks.

2. Bin Packing Problem (BPP)

Given a set of n items, each item j has a size of a_j, BPP aims to pack all items in the minimum number of identical sized bins without violating the capacity of bins (V). The problem can be mathematically formulated as follow:

$$\text{minimize } B = \sum_{i=1}^{n} y_i$$

$$\text{subject to } B \geq 1,$$

$$\sum_{j=1}^{n} a_j x_{ij} \leq V y_i, \forall i \in \{1, \ldots, n\}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad \forall j \in \{1, \ldots, n\}$$

$$y_i \in \{0, 1\}, \qquad \forall i \in \{1, \ldots, n\}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in \{1, \ldots, n\} \forall j \in \{1, \ldots, n\}$$

where $y_i = 1$ if bin i is used and $x_{ij} = 1$ if item j is put into bin i.

This mathematical formulation is generally NOT solvable by existing integer programming solvers like CPlex, Gurobi, LPSolve, especially when the number of items *n* is large. The solution space of bin packing problem is characterised by its huge size and plateau-like that makes it very challenging for traditional neighbourhood search methods. In order to consistently solve the problem with good quality solutions, metaheuristics and hyper-heuristics are used, which is the task of this coursework.

3. Problem instances

Over the years, a large number of BPP instances have been introduced by various research. See https://www.euro-online.org/websites/esicup/data-sets/ for a collection of different bin packing problem. In this coursework, we shall provide 3 instances files (binpack1.txt, binpack3.txt and binpack11.txt), respectively representing easy, medium and hard instances. From which 10 instances shall be selected for testing and evaluation of your algorithm in marking. For each test instance, only 1 run is executed, and its objective value is used for marking the performance component (see Section 5).

4. Experiments conditions and submission requirements

The following requirements should be satisfied by your program:

(1) You are required to submit **two files exactly**. The first file should contain all your program source codes. The second file is a coursework report. Please do **NOT** compress the files.

(2) Your source code should adopt a clean structure and be properly commented.

(3) Your report should include the followings:

- The main components of the algorithm, including solution encoding, fitness function, list of low-level heuristics as well as considerations regarding the intensification and diversification mechanisms. (**12 marks**).
- Statistical results (avg, best, worst of 5 runs) of the algorithm for all the problem instances, in comparison with the best published results (i.e. the absolute gap to the best results). **Note that** although your report should include results for 5 runs but your final submission should only have one single run for each instance (i.e. if you use the sketch code from the lab, set global variable NUM_OF_RUNS=1 before you submit the code). (**3 marks**)
- A short discussion/reflection on results and performance of the algorithm. (**5 marks**)

(4) Name your program file after your student id. For example, if your student number is 2019560, name your program as 2019560.c (or 2019560.cpp, or 2019560.py).

(5) Your program should compile and run without errors on either **CSLinux** Server or a computer in the IAMET406. Therefore, please fully tested 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 -lm 2019560.c -o 2019560
```

or

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

For Python programs, this second can be skipped.

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

```
./2019560 -s data_fle -o solution_file -t max_time
```

where `2019560` 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. In this coursework, maximum of 30 seconds is permitted. `soluton_file` is the file for output the best solutions by your algorithm. The format should be as follows:

```
# of problems
Instance_id1
obj=     objective_value  abs_gap
item_indx in bin0
item_indx in bin1
… …
Instance_id2
obj=     objective_value  abs_gap
item_indx in bin0
```

```
item_indx in bin1
 … …
```

An example solution file for problem data file "**binpack1.txt**" is available on moodle.

For submissions using Python, the compilation and running are combined in one command as follows:

```
python 2019560.py -s data_fle -o solution_file -t max_time
```

(7) The solution file output in (6) by your algorithm (`solution_file`) is expected to pass a solution checking test successfully using the following command on CSLInux:

```
./bpp_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 a command line message similar to: "Your total score out of 20 instances is: 80." If the solutions are infeasible for some instances, you would get error messages.

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

(8) Your algorithm should run only ONCE for each problem instance and each run should take no more than 30 seconds.

(9) Please carefully check the memory management in your program and test your algorithm with a full run on CSLinux (i.e. running multiple instances in one go). In the past, some submitted programs can run for 1-2 instances but then crashed because of *out-of-memory* error. This, if happens, will greatly affect your score.

(10) You must strictly follow policies and regulations related to Plagiarism. You are prohibited from using recent AI tools like ChatGPT/GPT-4 or other similar large language models (LLMs). Once a case is established, it will be treated as a plagiarism case and relevant policies and penalties shall be applied.

5. Marking criteria

- The quality of the experimental results (20 marks). Your algorithm shall be tested for a file containing 10 instances chosen from the provided set of instances. The performance of your algorithm is evaluated by computing the absolute gap with the best known results using

$$abs\_gap = your\_average\_objective - best\_known\_objective$$

Criteria	**Mark**
abs_gap < 0	**New best results! Bonus: 2 extra marks for each new best result.**
abs_gap <= 0	2 marks per instance
0<abs_gap <=1	1.5 marks per instance
1<abs_gap<=2	1 mark per instance
2< abs_gap <=3	0.5 mark per instance
• abs_gap >4 or • infeasible solution or • fail to output solution within required time limit	0 mark

- The quality of codes, including organisation of the functions/methods, naming conventions and clarity and succinctness of the comments (5 marks)
- Report (20 marks)

6. Submission deadline

3rd May 2024, 4pm Beijing Time

Standard penalties are applied for late submissions.

7. How to submit

Submit via Moodle.

8. Practical Hints

- Solution encoding for bin packing is slightly more challenging compared with knapsack program because both the number of bins to be used and the number of items to be packed in each bin are parts of decisions to be optimised. Therefore, the

data structure that is used to hold the packing information cannot be implemented via fixed-size arrays. You may consider to use **vector** from C++ STL (standard template library) which requires you to include <vector.h> as header file. If you prefer C style without classes, the following data type would be also acceptable:

```
struct bin_struct {
   std::vector<item_struct> packed_items;
   int cap_left;
};
struct solution_struct {
   struct problem_struct* prob; //maintain a shallow copy of problem data
   float objective;
   int feasibility; //indicate the feasibility of the solution
   std::vector<bin_struct> bins;
};
```

In this way, you could open/close bins and at the same time to add/remove items for a specific bin through API functions provided by the vector library.

- The search space of bin packing problem has a lot of plateaus that make the problem extremely difficult for simple neighbourhood methods. Therefore, multiple low-level heuristics are suggested within a perturbative hyper-heuristic method. You are free to select any of the perturbative hyper-heuristic methods described in (https://link.springer.com/article/10.1007/s10288-011-0182-8), as well as some of the more recent ones (https://www.sciencedirect.com/science/article/pii/S0377221719306526).
- Your algorithm must be runnable on CSLinux and/or computers on IAMET406. Therefore, you are not permitted to use external libraries designed specifically for optimisation.