Documentation

Office Space Allocation Library (OSAL)

D.1 Introduction

This Read Me file is created to help the user execute the baseline heuristics present in this library. This documentation combined with the dissertation report and the commented code will provide the users with a thorough explanation of the whole project. A complete run of this library is characterized by 4 stages:

1. Input Functionality
2. Initialisation (Constructive Heuristics)
3. Neighbourhood Exploration (Driving Metaheuristics)
4. Output Functionality

The aim of this library is to provide the user with several heuristics which he/she can utilize to solve the Office Space Allocation (OSA) problem. "Solving" here means getting an entity-room mapping at the end of all four stages. This entity-room mapping is known as a solution or allocation. The heuristics implemented here are taken from (Landa-Silva, 2003). The test instances used here are taken from (Landa-Silva, 2003) and (Ulker and Landa-Silva, 2011).

D.2 Files

This code repository has the following files.

Test Instances (csv folder)

- nott.txt
- p000_n025.txt
- p025_n000.txt
- s000_v000.txt
- s000_v100.txt
- s100_v000.txt
- s100_v100.txt

Stage 1: Input Functionality

- Input.py

Stage 2: Initialisation (Constructive Heuristics)

- AllocateRnd_Rnd.py
- AllocateRnd_BestRnd.py
- AllocateWgt_BestRnd.py
- AllocateCsrt_BestRnd.py
- AllocateBestAll.py

Stage 3: Neighbourhood Exploration (Driving Metaheuristics)

- IIARnd_Rnd.py
- IIARnd_BestRnd.py
- SAARnd_Rnd.py
- SAARnd_BestRnd.py

Stage 4: Output Functionality

- Output.py

Example Program

- Example.py
- Example Output.txt

D.3 Building a Complete Program

To create a complete program, code from one python file from each stage has to be taken. The Input.py and Output.py file is selected by default. Code from these files are copied and pasted in an empty Python file. Now this file can be executed to find solutions for the OSA problem.

IMPORTANT: Please ensure that the csv folder is in the same folder as the empty python file that the user pastes the code in.

IMPORTANT: In Python, a function has to be defined before it is called and hence the following order has to be maintained when copying and pasting code in a new Python file: Input.py code followed by Output.py code followed by Constructive Heuristic code followed by Driving Metaheuristic code. This is because both the constructive heuristic and driving metaheuristic codes call the functions present in Output.py. Another important point to remember is that the flow of control will still follow the order highlighted in Section 1.1.

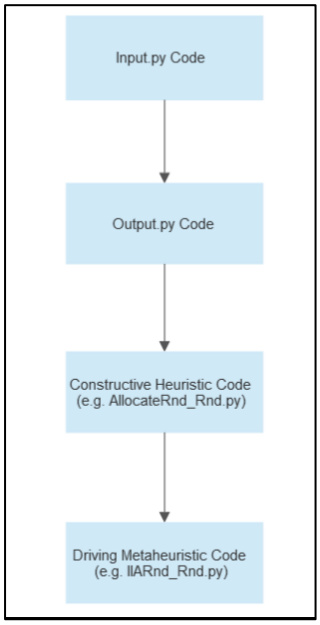

Figure D.1 Flow of Control in program

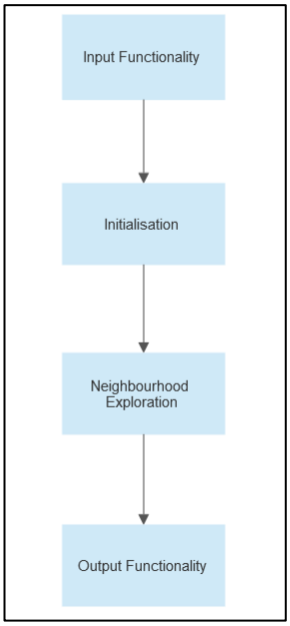

Figure D.2 Order of copying and pasting code in an empty Python file

IMPORTANT: An example complete program is included in this code repository.

It is titled Example.py. It is created by combining code from Input.py, Output.py, AllocateRnd_Rnd.py and IIARnd_Rnd.py in that order. An example output text file (Example Output.txt) is also included to show the users how the output looks after executing such complete programs.

D.4 Stage 1: Input Functionality (Input.py)

The following flowchart explains the input process in detail:

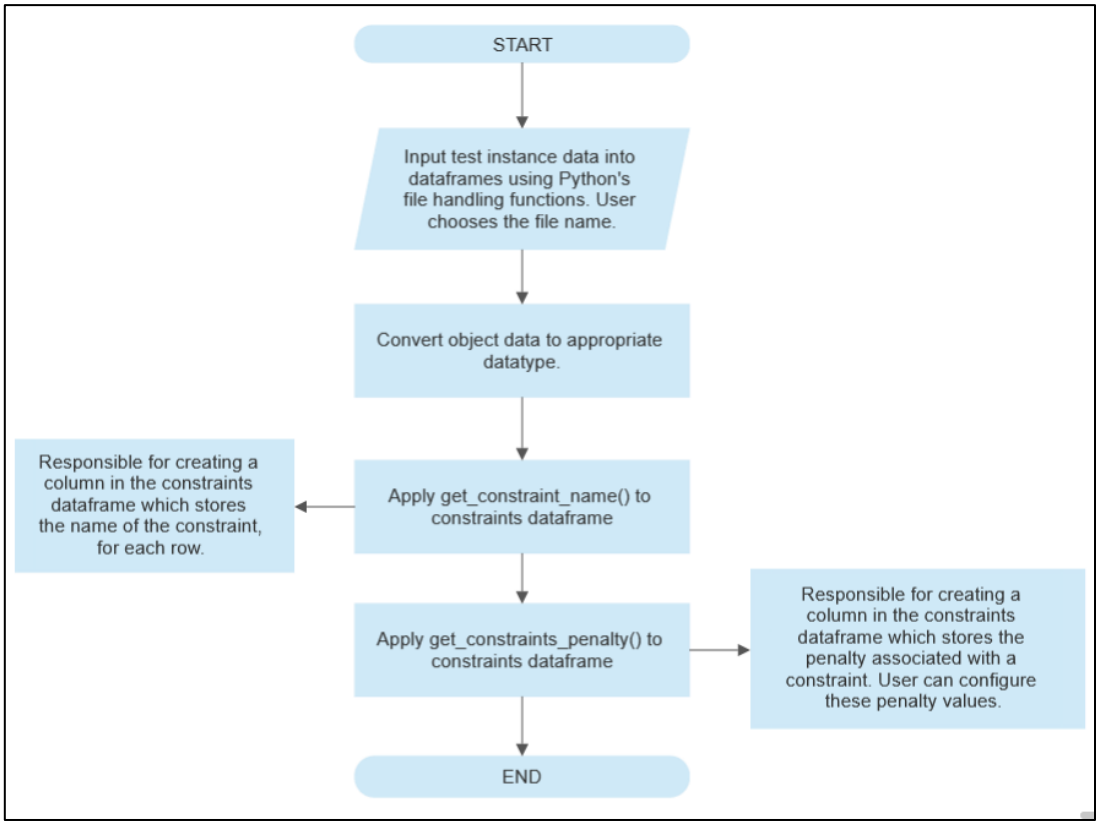

Figure D.3 Input Functionality

The following characteristics were kept in mind while creating this functionality:

1. File Format: The test instances are in text format.
2. Delimiters: There are spaces between columns to denote separation between attributes.
3. Headers : Although section headers are present, no column headers are present. Hence they were manually added in afterwards.
4. Data Types: Python reads all information in and gives it the object datatype. Hence object to float datatype conversion had to be carried out.
5. Additional Data: Data regarding constraint penalty is added to the Constraints Dataframe.

Refer Section 4.4 in the Dissertation Report for more details.

D.5 Stage 2: Constructive Heuristics

D.5.1 AllocateRnd-Rnd (AllocateRnd_Rnd.py)

The following flowchart explains the AllocateRnd-Rnd constructive heuristic in detail:

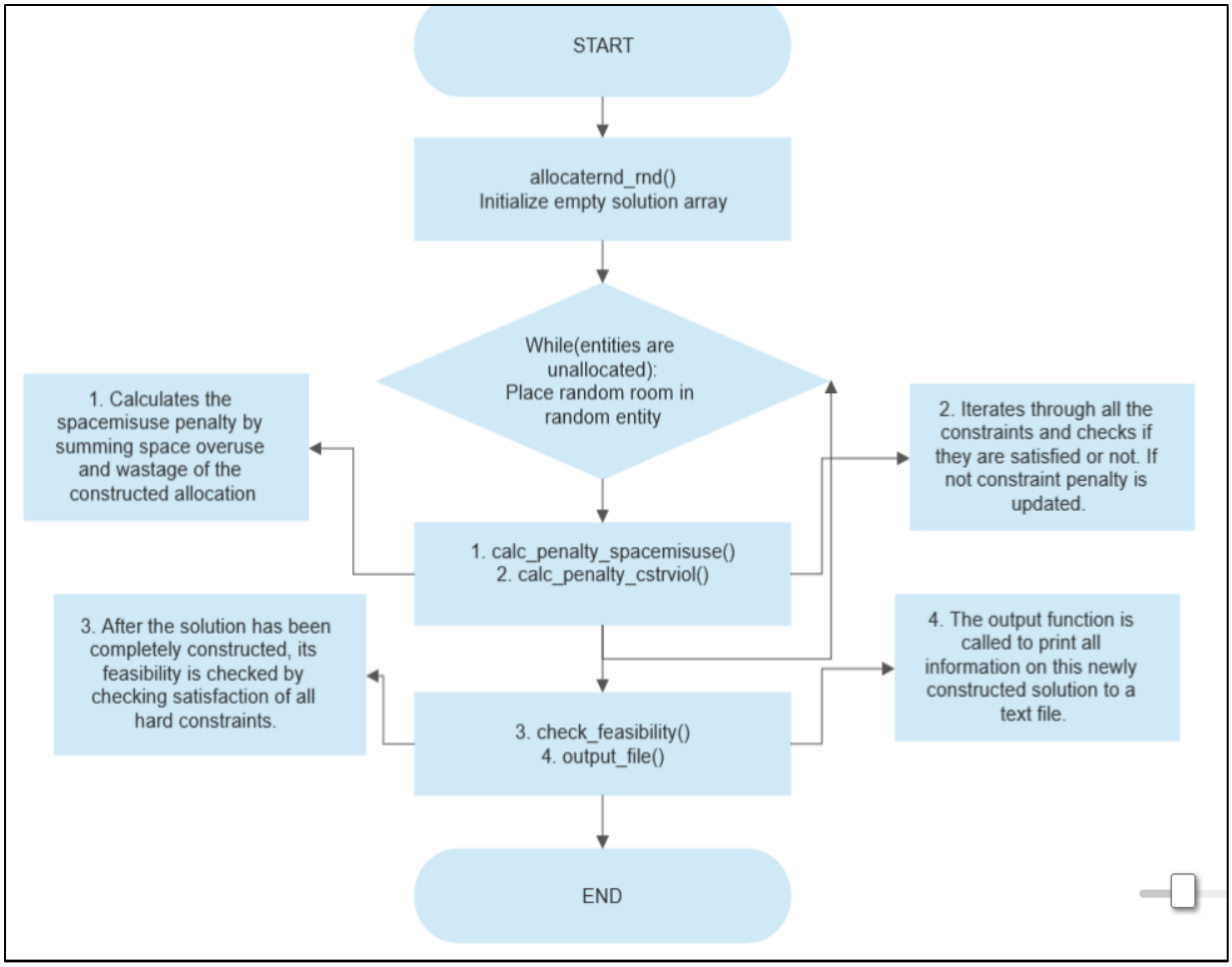

Figure D.4 AllocateRnd-Rnd

Refer Section 4.5 in the Dissertation Report for more details.

D.5.2 AllocateRnd-BestRnd (AllocateRnd_BestRnd.py)

The following flowchart explains the AllocateRnd-BestRnd constructive heuristic in detail:

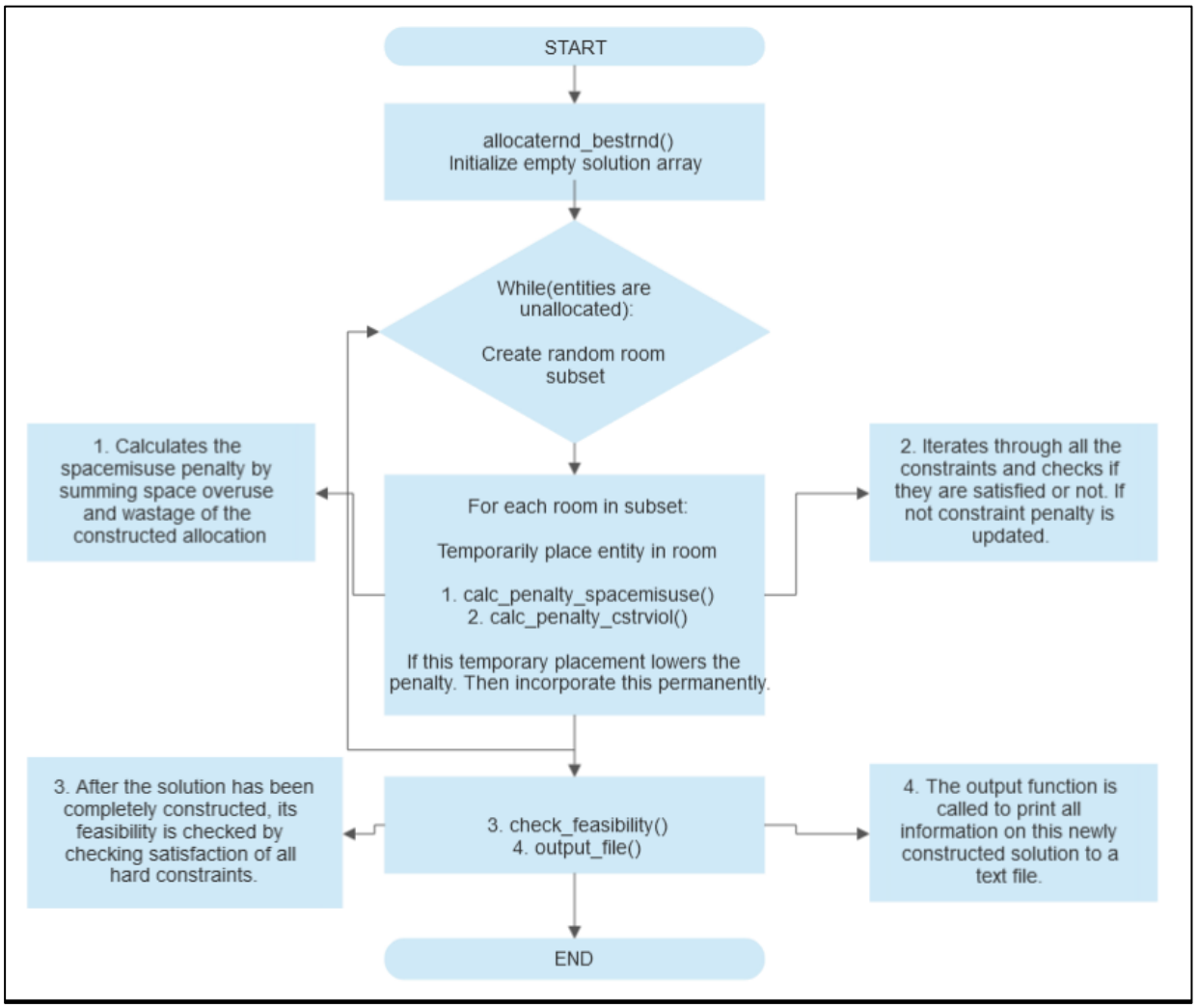

Figure D.5 AllocateRnd-BestRnd

Refer Section 4.5 in the Dissertation Report for more details.

D.5.3 AllocateWgt-BestRnd (AllocateWgt_BestRnd.py)

The following flowchart explains the AllocateWgt-BestRnd constructive heuristic in detail:

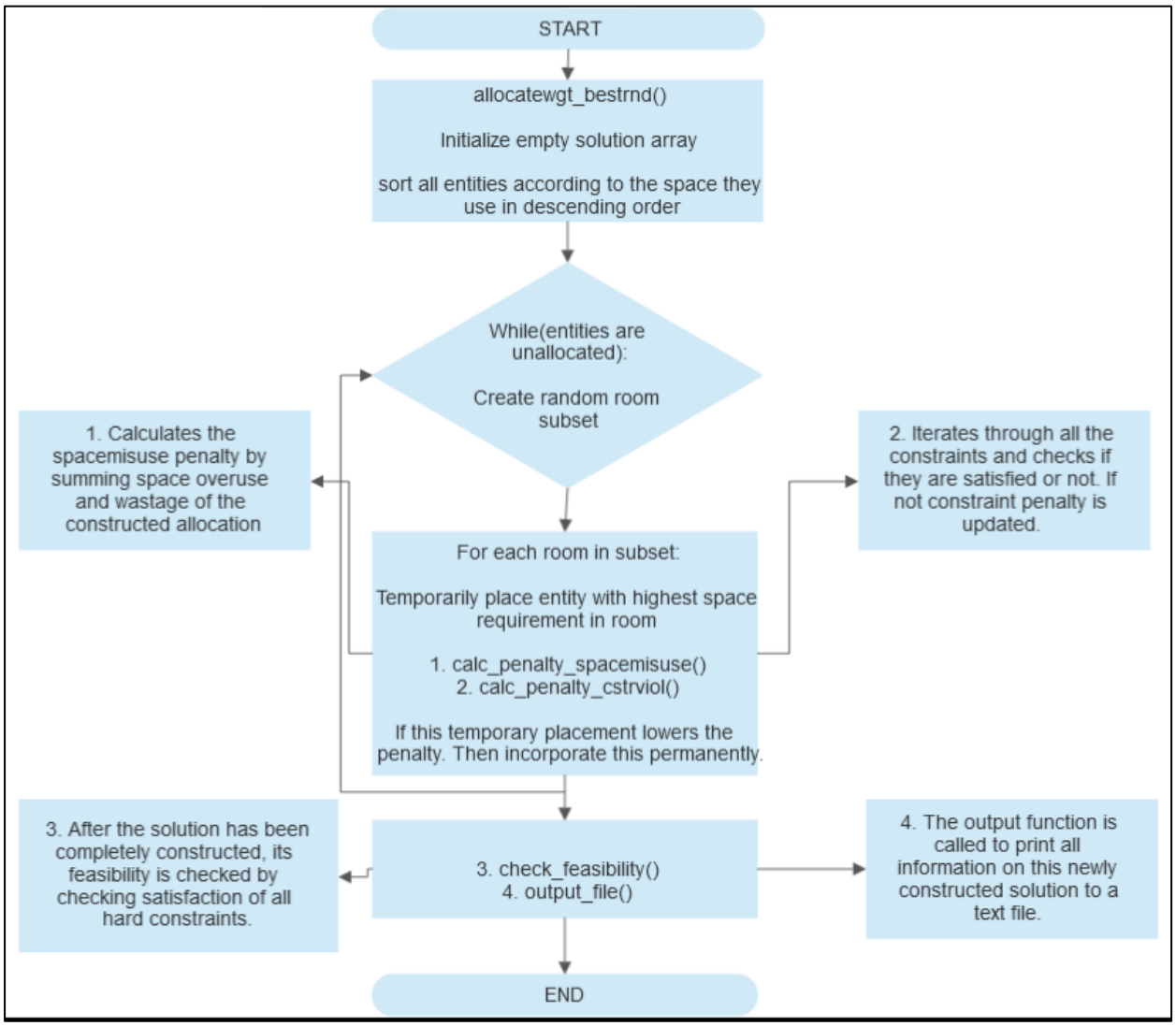

Figure D.6 AllocateWgt-BestRnd

Refer Section 4.5 in the Dissertation Report for more details.

D.5.4 AllocateBestAll (AllocateBestAll.py)

The following flowchart explains the AllocateBestAll constructive heuristic in detail:

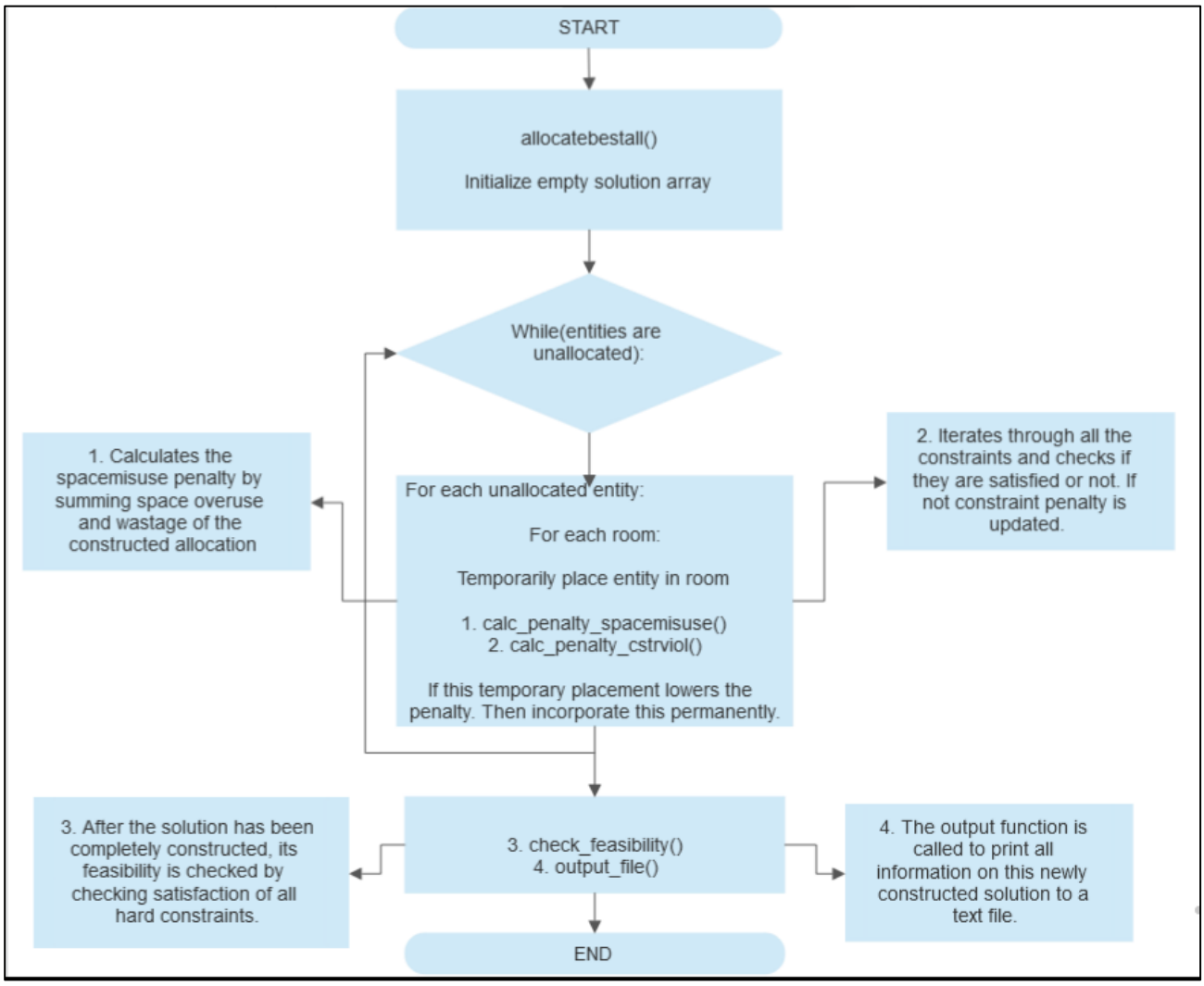

Figure D.7 AllocateBestAll

Refer Section 4.5 in the Dissertation Report for more details.

D.5.5 AllocateCstr-BestRnd (AllocateCstr_BestRnd.py)

The following flowchart explains the AllocateCstr-BestRnd constructive heuristic in detail:

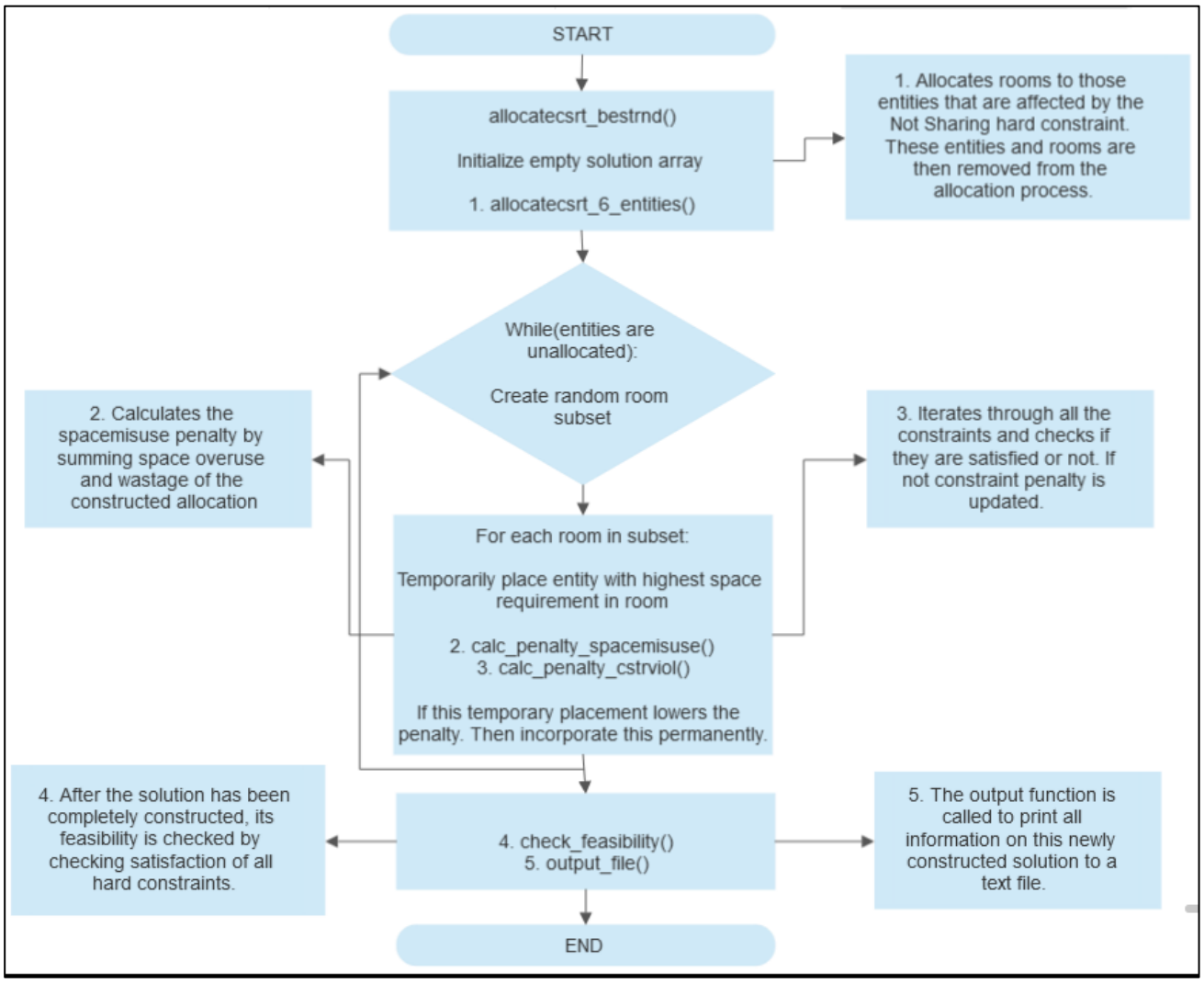

Figure D.8 AllocateCstr-BestRnd

Refer Section 4.5 in the Dissertation Report for more details.

D.6 Stage 3: Neighbourhood Exploration Metaheuristics

D.6.1 IIARnd-Rnd (IIARnd_Rnd.py)

The following flowcharts explain the IIARnd-Rnd driving metaheuristic in detail:

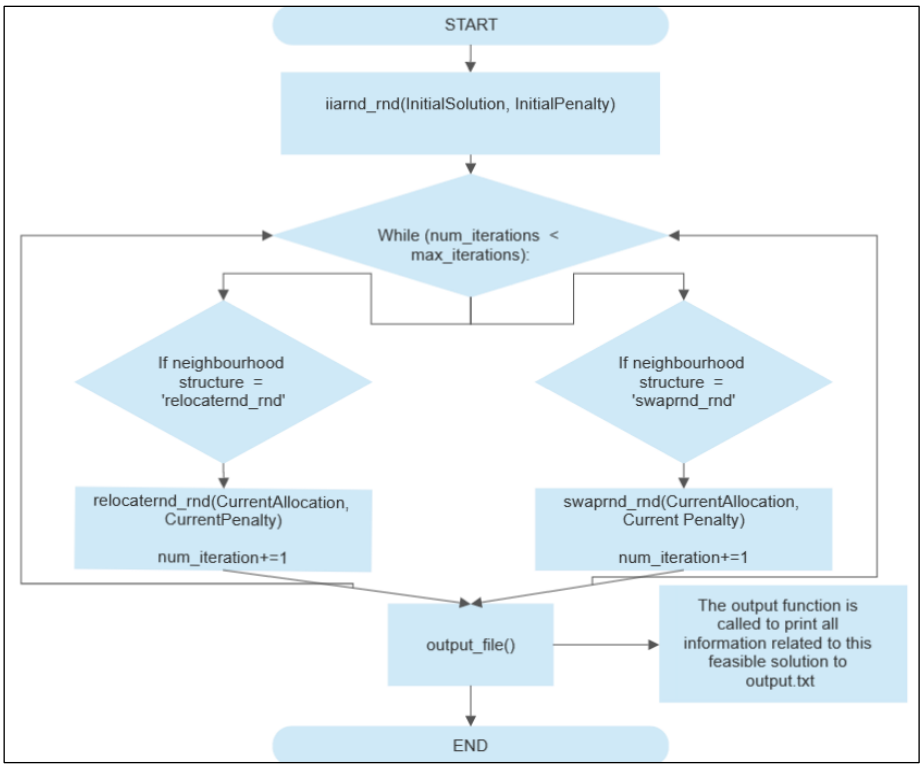

Figure D.9 IIARnd-Rnd main function

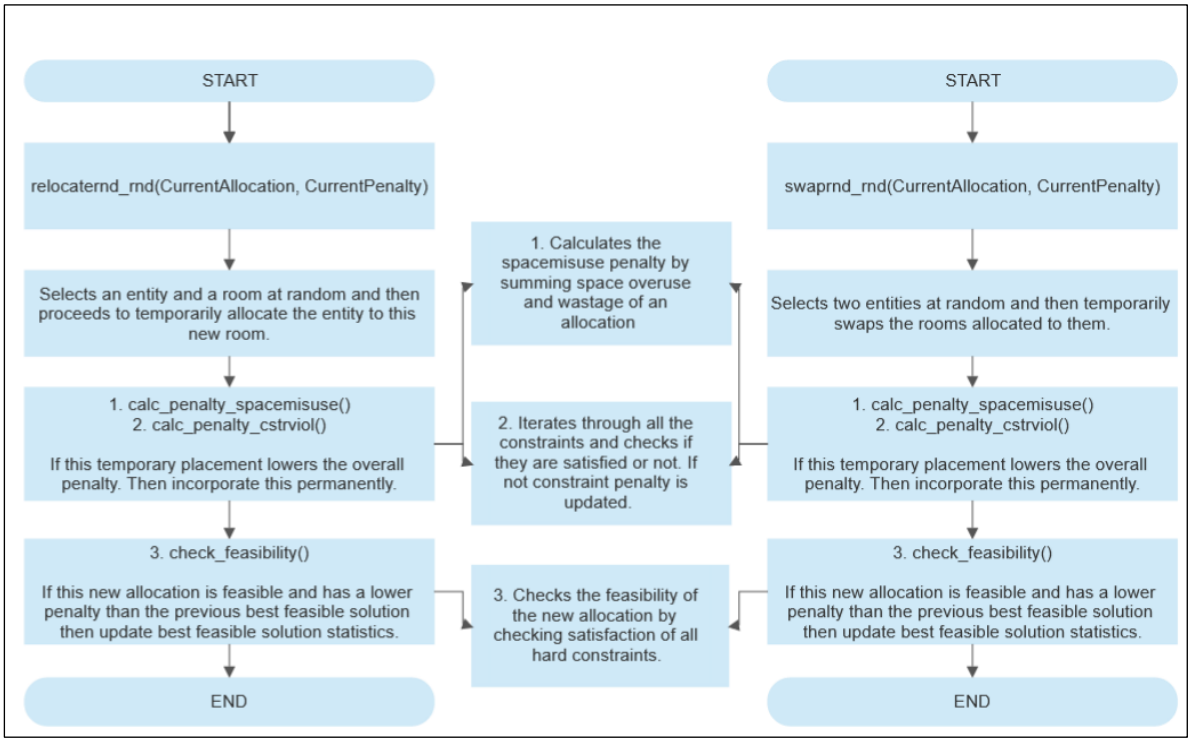

Figure D.10 Working of relocaternd_rnd() and swaprnd_rnd in IIARnd-Rnd

Refer Section 4.6 and 4.7.1.1 in the Dissertation Report for more details.

D.6.2 SAARnd-Rnd (SAARnd_Rnd.py)

The following flowcharts explain the SAARnd-Rnd driving metaheuristic in detail:

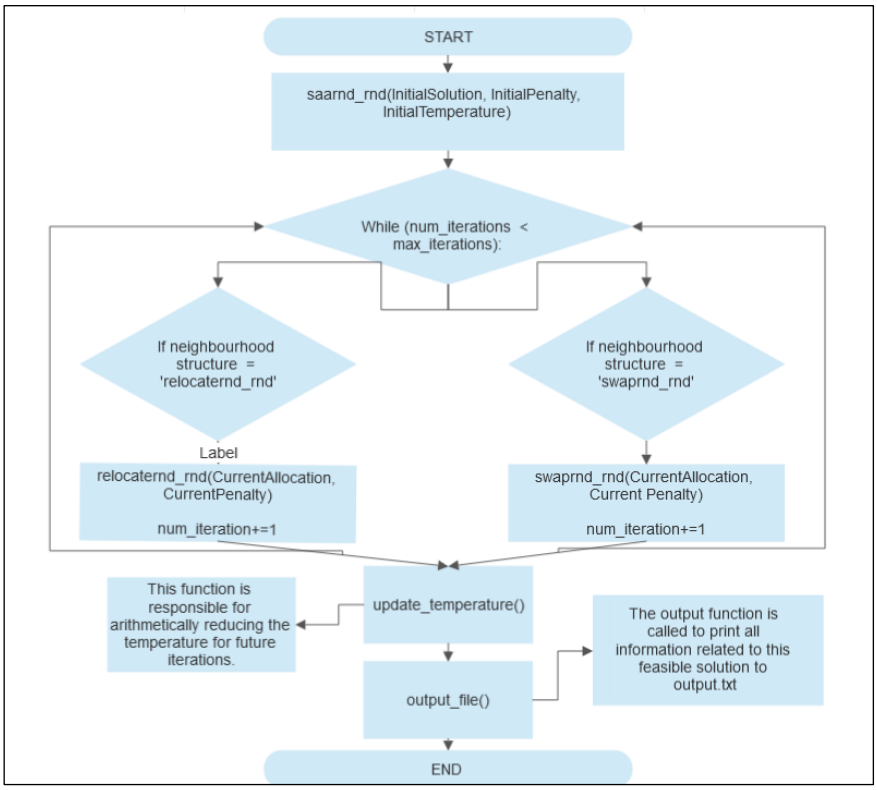

Figure D.11 SAARnd-Rnd main function

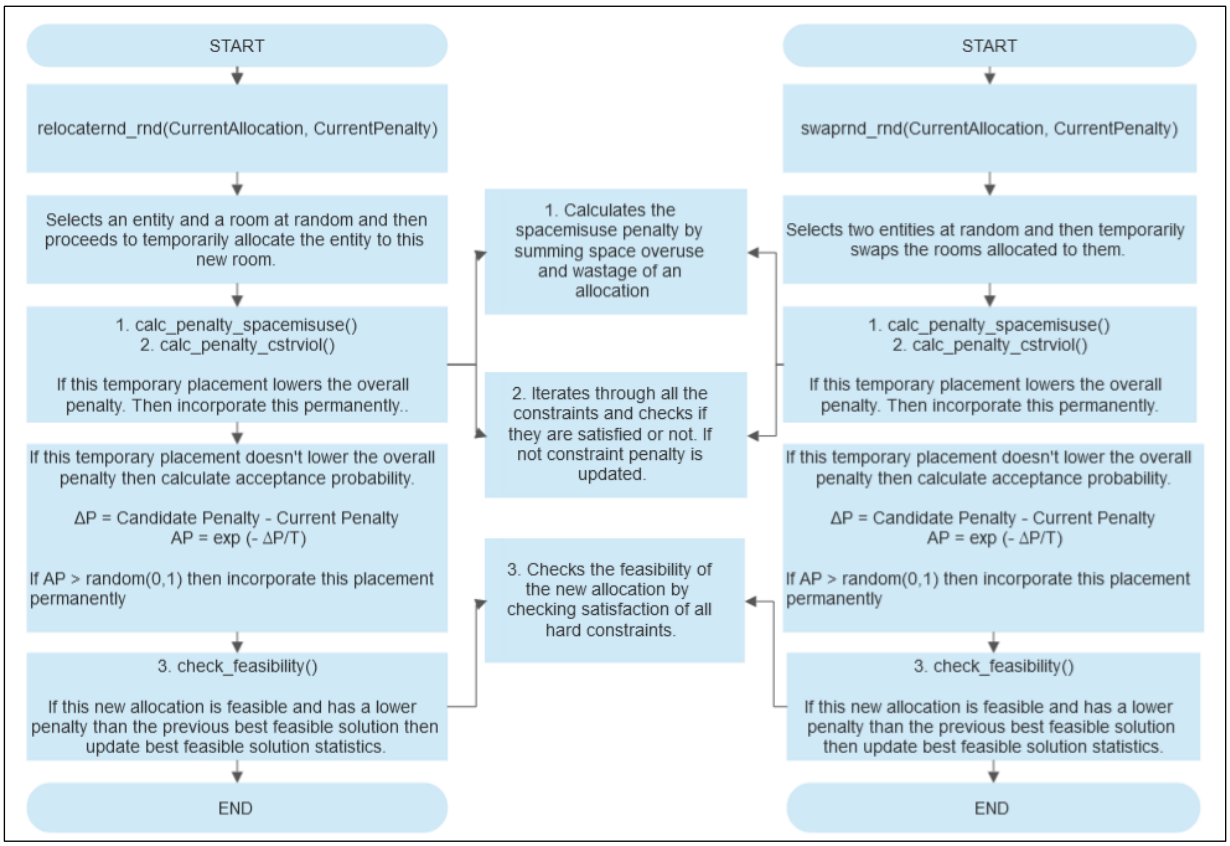

Figure D.12 Working of relocaternd_rnd() and swaprnd_rnd() in SAARnd_Rnd

Refer Section 4.6 and 4.7.2.1 in the Dissertation Report for more details.

D.6.3 IIARnd-BestRnd (IIARnd_BestRnd.py)

The following flowcharts explain the IIARnd-BestRnd driving metaheuristic in detail:

(Refer Section 4.6 and 4.7.1.2 in the Dissertation Report for more details.)

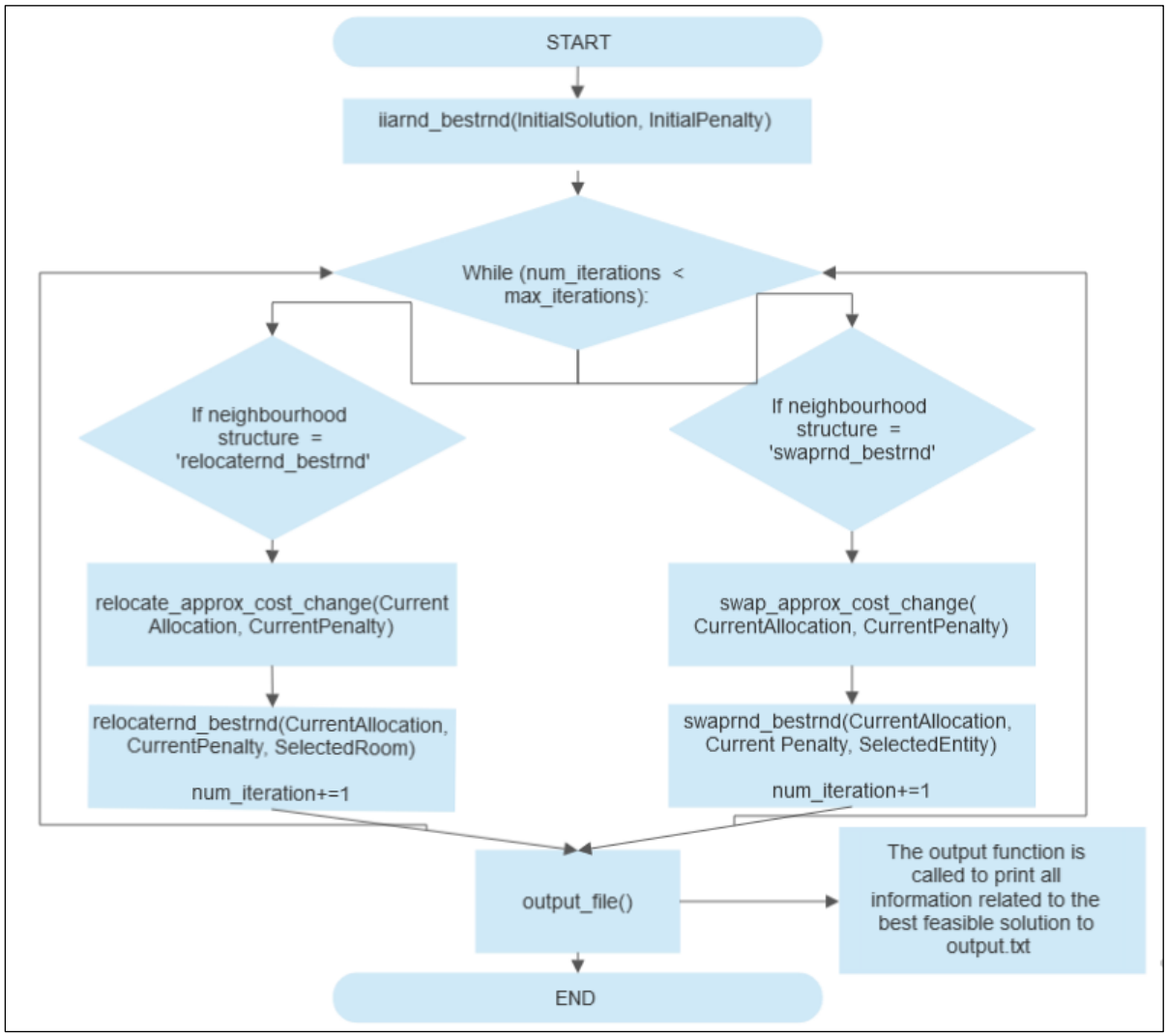

Figure D.13 IIARnd-BestRnd main function

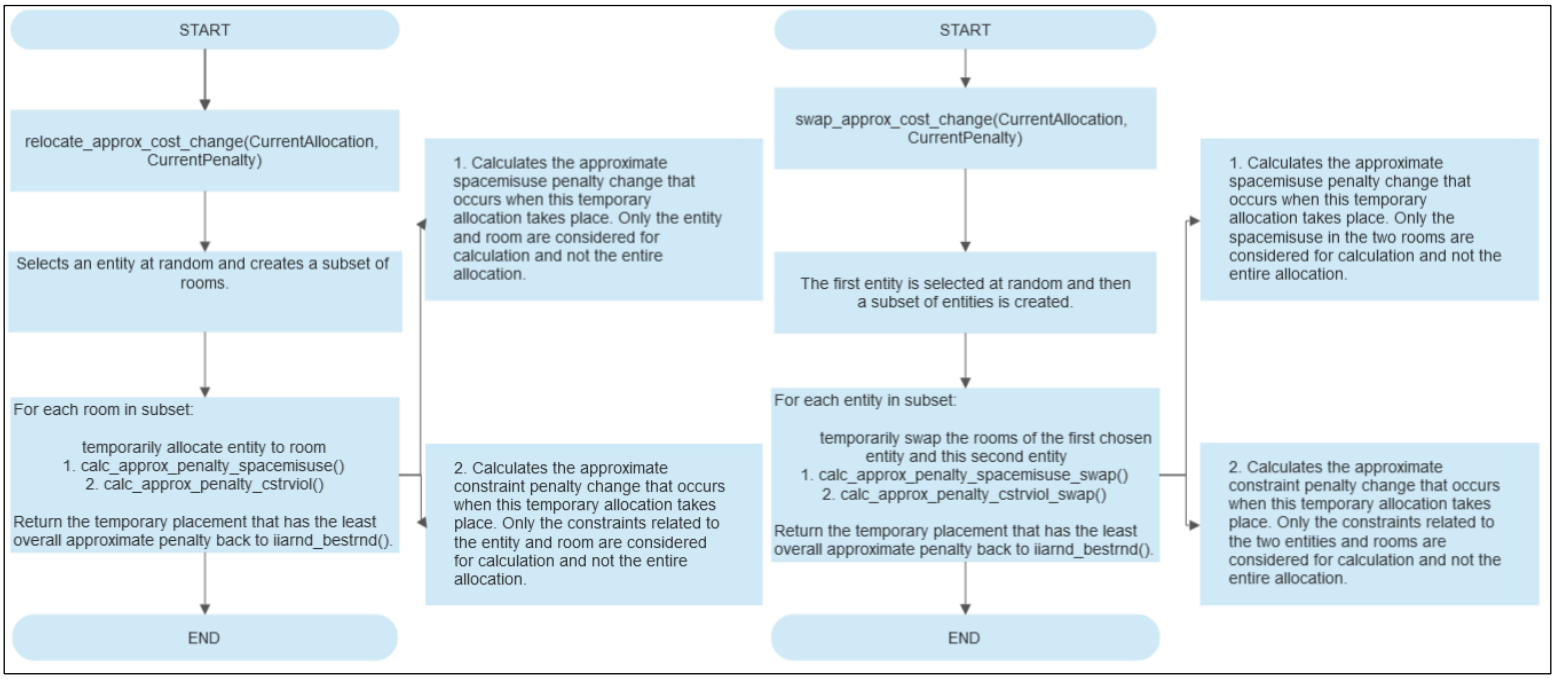

Figure D.14 Working of relocate_approx_cost_change() and swap_approx_cost_change() in IIARnd-BestRnd

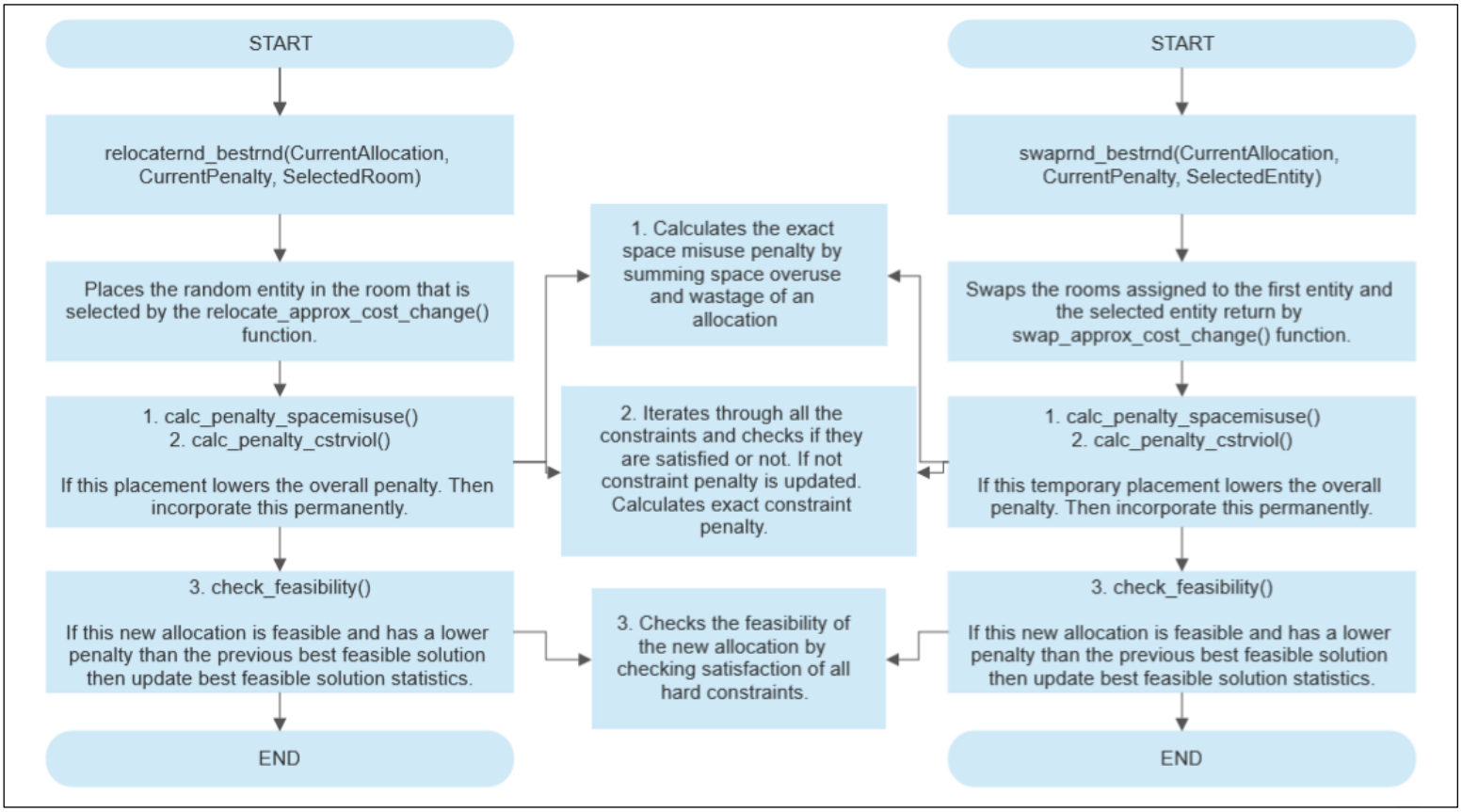

Figure D.15 Working of relocaternd_bestrnd() and swaprnd_bestrnd() in IIARnd-BestRnd

D.6.4 SAARnd-BestRnd (SAARnd_BestRnd.py)

The following flowcharts explain the SAARnd-BestRnd driving metaheuristic in detail:

(Refer Section 4.6 and 4.7.2.2 in the Dissertation Report for more details.)

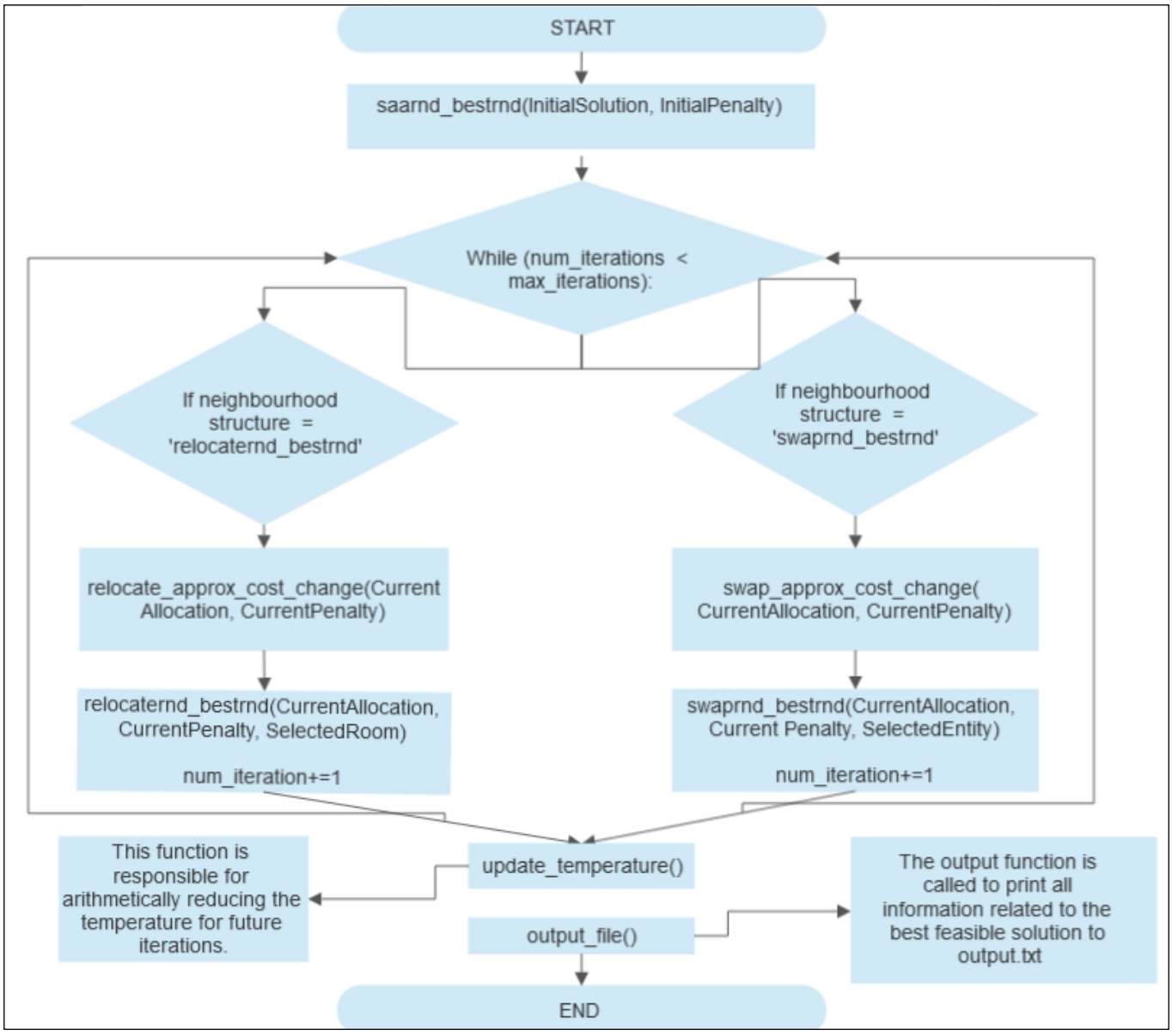

Figure D.16 SAARnd-BestRnd main function

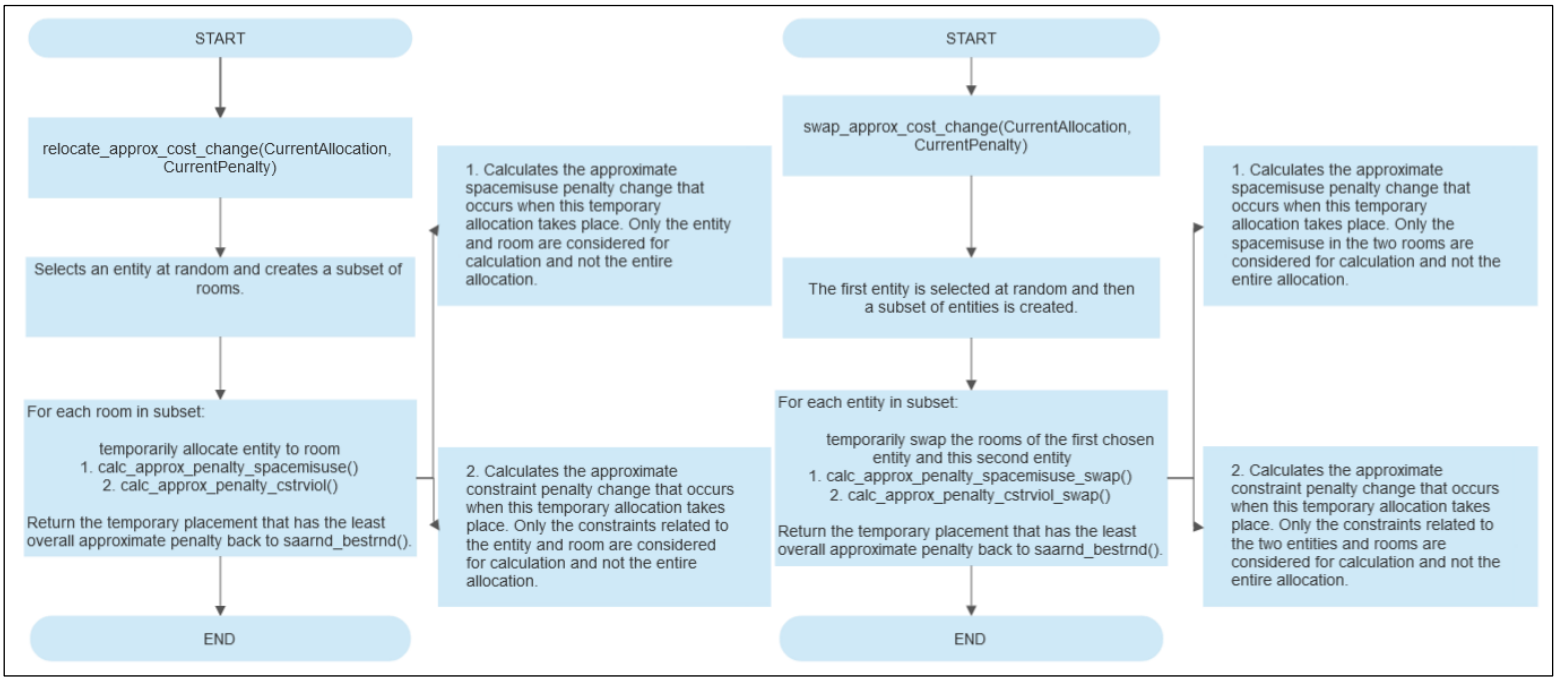

Figure D.17 Working of relocate_approx_cost_change() and swap_approx_cost_change() in SAARnd-BestRnd

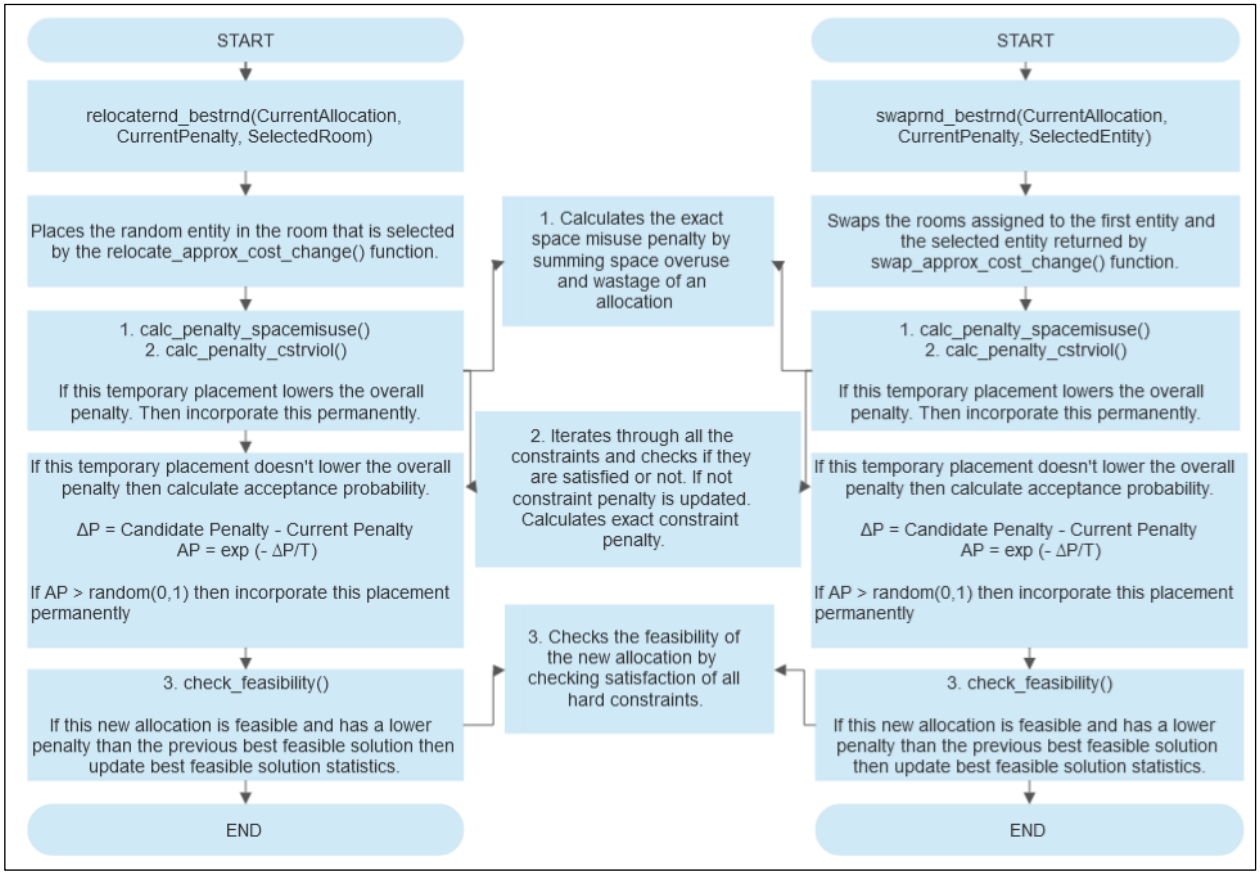

Figure D.18 Working of relocaternd_bestrnd() and swaprnd_bestrnd() in SAARnd-BestRnd

D.7 Stage 4: Output Functionality (Output.py)

The following flowchart explains the output functionality in detail:

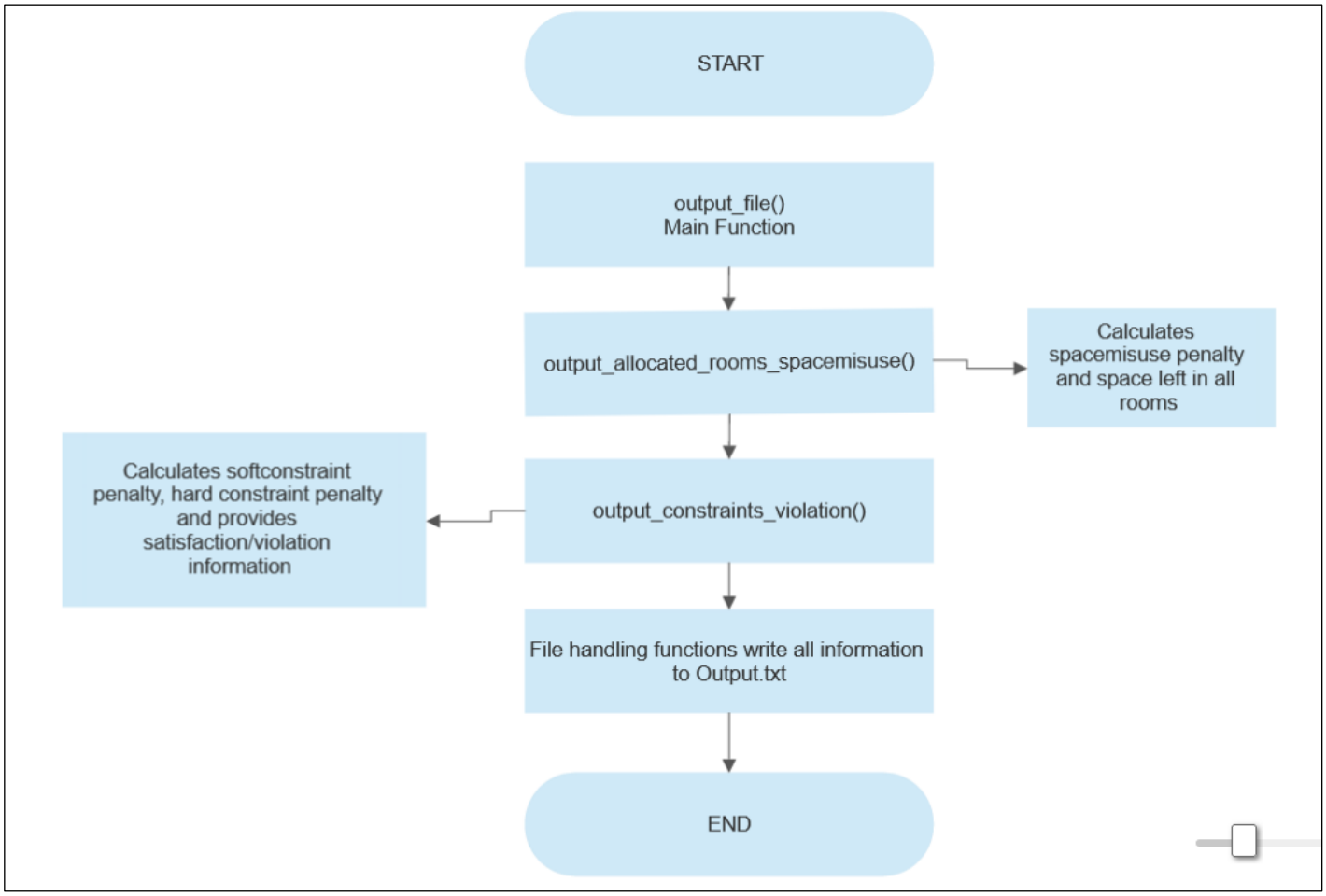

Figure D.19 Output Functionality

The following information on the best feasible allocation is pushed to Output.txt:

1. The iteration number at which the best feasible allocation was reached.
2. The values of the different penalty components for the recorded best feasible solution i.e. total penalty, soft constraint violation penalty and space misuse penalty.
3. The Entity-Room mapping of the recorded best feasible solution.
4. The amount of space utilized and left for each room in the dataset.
5. The satisfaction/violation status for each constraint in the dataset.

IMPORTANT: Renaming the Output.txt file generated by the end of the execution is required if the user wants to permanently save that file. This is because the contents of Output.txt will be replaced if the user executes the program again.

References

Landa Silva, J.D. (2003). Metaheuristic and multiobjective approaches for space allocation (Doctoral dissertation, University of Nottingham).

Ülker, Ö. and Landa-Silva, D. (2011). Designing difficult office space allocation problem instances with mathematical programming. In Experimental Algorithms: 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings 10 (pp. 280-291). Springer Berlin Heidelberg