Faculdade de Engenharia da Universidade do Porto
Heuristics and Metaheuristics
1st Semester
2020/2021

# Heuristics for the Set Covering Problem

Tiago Filipe Sousa Gonçalves

November 4, 2020

## Contents

# 1 Introduction

Under the scope of the curricular unit "Heuristics and Metaheuristics", the students were proposed a three-assignment project based on the set covering problem instances proposed by [1]. The first phase consisted on the study and proposal of different constructive heuristics algorithms (see section 2), the second phase consisted on the study and proposal of different improvement heuristics (see section 3) and the third phase consisted on the study and implementation of different local-based search metaheuristics (see section 4). All the data, code, results and reports can be found in public GitHub repository*. The algorithms were implemented in Python and the statistical analysis was done in Microsoft Excel using the add-on "Real Statistics Using Excel" †.

# 2 Constructive Heuristics

## 2.1 General Aspects

Before the explanation of the algorithms, there are general implementation details that deserve to be presented, since they are common elements in every constructive heuristics algorithm.

### 2.1.1 Solution Representation

For the purpose of this assignment, the *Solution Representation* is an array that contains the sets (columns indices) needed to cover the entire universe of elements (rows): $[\text{Column}_i, ..., \text{Column}_n]$ for $i \in [0, n)$ if $\text{Column}_i \in$ Solution, where n = number of sets of the set covering instance. Please note that since this project was implemented in Python language, the column indices were processed to start in 0.

### 2.1.2 Monitored Variables

During the execution of the constructive heuristics algorithms, the following variables were monitored:

- *Rows already covered*: the list of elements that our current solution already covers;
- *Candidate columns to be added*: the list of sets that are still available to be added to the solution;
- *Number of columns that cover each row*: the frequency of sets per element;
- *Current solution*: the sets that we added to our solution in a specific iteration;
- *Current cost*: the total cost of our solution in a specific iteration.

### 2.1.3 Set Redundancy Removal

The *Set Redundancy Removal* procedure is done as follows:

**Algorithm 1:** Set Redundancy Removal.

**Result:** The processed solution, with less redundant sets (columns).
Start from the current solution, obtained after the execution of a constructive heuristics algorithm;
**while** number of rows already covered < number of rows to be covered **do**
    check the set (column) which covers more rows;
    check if this set (column) contains other set(s);
    **if** this set contains other set(s): **then** remove the other (smaller) sets from the current solution ;
**end**

*Available at https://github.com/TiagoFilipeSousaGoncalves/pdeec-hm-assignments
†Available at https://www.real-statistics.com

#### 2.1.4 Cost Effectiveness

The *Cost Effectiveness* was used in some of the constructive heuristics algorithms, thus, it should be explained in this report. It is defined as follows:

$$\alpha = \frac{\text{Cost}(S)}{|S - X|} \tag{1}$$

where, $\alpha$ is the *Cost Effectiveness*, $\text{Cost}(S)$ is the cost of the candidate set $S$, $X$ is the set of elements covered by the current solution and $|S - X|$ is the difference between the elements covered by set S and the elements covered by the current solution $X$.

### 2.2 Constructive Heuristics Algorithms

Under the scope of this assignment, five different constructive algorithms were developed: Constructive Heuristics Algorithm #1 (CH1), Constructive Heuristics Algorithm #2 (CH2), Constructive Heuristics Algorithm #3 (CH3), Constructive Heuristics Algorithm #4 (CH4), and Constructive Heuristics Algorithm #5 (CH5). However, since the main objective of this assignment was to propose, study and test three of them, the author decided to present three of them in this report (data and statistical analysis for all of them are available in the GitHub repository, presented in section 1). Each algorithm was written as a function that returns the solution, the cost and if post-processing is enabled, the processed solution and the processed cost. All the non-processed and processed solutions and costs were saved into files for further use.

#### 2.2.1 Constructive Heuristics Algorithm #1 (CH1)

Constructive Heuristics Algorithm #1 (CH1) is implemented as follows:

**Algorithm 2:** CH1 Algorithm.

```
Result: CH1 Algorithm solution.
Start empty solution;
while number of rows already covered < number of rows to be covered do
    pick a random element (row) to be covered;
    check all the sets (columns) that contain this element (row);
    for each of these possible sets (columns) do compute the set cost-effectiveness;
    pick the set with lower cost-effectiveness value and add it to the solution;
    check the rows that are covered by this set and update the number of rows already covered
end
if post-processing is enabled then
    apply set redundancy removal procedure
end
```

#### 2.2.2 Constructive Heuristics Algorithm #2 (CH2)

Constructive Heuristics Algorithm #2 (CH2) is implemented as follows:

**Algorithm 3:** CH2 Algorithm.

```
Result: CH2 Algorithm solution.
Start empty solution;
while number of rows already covered < number of rows to be covered do
    check all the sets (columns) that are available to be added;
    for each of these possible sets (columns) do compute the set cost-effectiveness;
    pick the set with lower cost-effectiveness value and add it to the solution;
    check the rows that are covered by this set and update the number of rows already covered;
end
if post-processing is enabled then
    apply set redundancy removal procedure;
end
```

#### 2.2.3 Constructive Heuristics Algorithm #5 (CH5)

Constructive Heuristics Algorithm #5 (CH5) is implemented as follows:

**Algorithm 4:** CH5 Algorithm.

**Result:** CH5 Algorithm solution.
Check the element (row) with higher frequency, $f$;
Solve the linear-programming problem for the set covering problem and obtain $x_j^*, j \in [0, n)$;
Start empty solution;
**while** number of rows already covered < number of rows to be covered **do**
  **for** each $x_j^* > \frac{1}{f}$ **do** add $x_j^*$ to the solution;
  check the rows that are covered by this set and update the number of rows already covered;
**end**
**if** post-processing is enabled **then**
  apply set redundancy removal procedure;
**end**

# 3 Improvement Heuristics

## 3.1 Improvement Heuristics Algorithms

The goal of the second assignment was to propose, study and test, at least, two types of improvement heuristics: first-neighbour improvement heuristics (FI) and best-neighbour improvement heuristics (BI). Besides those two types, the author implemented two more, a hybrid approach and the algorithm proposed by [2], which are also present in this report. In total, the author studied four improvement heuristics algorithms: Improvement Heuristics Algorithm #1 (IH1), Improvement Heuristics Algorithm #2 (IH2), Improvement Heuristics Algorithm #3 (IH3), and Improvement Heuristics Algorithm #4 (IH4). Under the main purpose of this report, the study of the impact of these local-search algorithms is presented for the constructive algorithms CH1, CH2 and CH5 (data and statistical analysis for all of them are available in the GitHub repository, presented in section 1). Each algorithm was written as a function that returns the initial solution, the initial cost, the final solution, the final cost and the history (which contains the cost of the current solution's neighbours in each iteration).

### 3.1.1 Improvement Heuristics Algorithm #1 (IH1)

Improvement Heuristics Algorithm #1 (IH1) was implemented as follows:

**Algorithm 5:** IH1 Algorithm (FI).

```
Result: IH1 Algorithm solution.
Load current solution;
Compute current solution cost (current cost);
while current iteration ≤ maximum number of iterations and current patience value ≤ maximum
 patience value do
    valid neighbour found = False;
    while valid neighbour found not True do
        randomly choice a set (column) of the current solution;
        if the removal of this set of the current solution does not jeopardise the universality of the
          solution then the neighbour solution is automatically obtained through the removal of this set;
        valid neighbour found = True ;
        else search for sets that can be swapped with this one;
        build a list of the neighbours of the solution which are obtained through the swap movement
          (remove this set and insert a new one);
        valid neighbour found = True ;
    end
    randomly choose a neighbour from the list of neighbours;
    compute the cost of the selected neighbour (new cost);
    if the new cost < the current cost then the current cost = the new cost;
    the current solution = selected neighbour;
    current iteration+ = 1;
    current patience value = 0 ;
    else current iteration+ = 1;
    current patience value+ = 1 ;
end
```

### 3.1.2 Improvement Heuristics Algorithm #2 (IH2)

Improvement Heuristics Algorithm #2 (IH2) was implemented as follows:

**Algorithm 6:** IH2 Algorithm (BI).

```
Result: IH2 Algorithm solution.
Load current solution;
Compute current solution cost (current cost);
while current iteration ≤ maximum number of iterations and current patience value ≤ maximum
 patience value do
    valid neighbour found = False;
    while valid neighbour found not True do
        randomly choice a set (column) of the current solution;
        if the removal of this set of the current solution does not jeopardise the universality of the
         solution then the neighbour solution is automatically obtained through the removal of this set;
        valid neighbour found = True ;
        else search for sets that can be swapped with this one;
        build a list of the neighbours of the solution which are obtained through the swap movement
         (remove this set and insert a new one);
        valid neighbour found = True ;
    end
    compute the cost of all the neighbours in the list;
    choose the neighbour (selected neighbour) that corresponds to the lower cost (new cost);
    if the new cost < the current cost then the current cost = the new cost;
    the current solution = selected neighbour;
    current iteration+ = 1;
    current patience value = 0 ;
    else current iteration+ = 1;
    current patience value+ = 1 ;
end
```

### 3.1.3 Improvement Heuristics Algorithm #3 (IH3)

Improvement Heuristics Algorithm #3 (IH3) was implemented as follows:

**Algorithm 7:** IH3 Algorithm (Tentative Hybrid Approach).

```
Result: IH3 Algorithm solution.
Load current solution;
Compute current solution cost (current cost);
Initialise empty list of chosen sets (chosen sets list);
while current iteration ≤ maximum number of iterations and current patience value ≤ maximum
 patience value do
    valid neighbour found = False;
    while valid neighbour found not True do
        high cost set found = False while valid high cost set found not True do
            choose the set with higher cost from the current solution;
            if this set is not in the chosen sets list then add this set to the chosen sets list;
            high cost set found = True ;
            else
                if all the sets are in the chosen sets list then reboot chosen sets list as an empty list ;
                else choose the next set with higher cost from the current solution;
                add this set to the chosen sets list;
                high cost set found = True ;
            end
        end
        if the removal of this set of the current solution does not jeopardise the universality of the
         solution then the neighbour solution is automatically obtained through the removal of this set;
        valid neighbour found = True ;
        else search for sets that can be swapped with this one;
        build a list of the neighbours of the solution which are obtained through the swap movement
         (remove this set and insert a new one);
        valid neighbour found = True ;
    end
    compute the cost of all the neighbours in the list;
    choose the neighbour (selected neighbour) that corresponds to the lower cost (new cost);
    if the new cost < the current cost then the current cost = the new cost;
    the current solution = selected neighbour;
    current iteration+ = 1;
    current patience value = 0 ;
    else current iteration+ = 1;
    current patience value+ = 1 ;
end
```

### 3.1.4 Improvement Heuristics Algorithm #4 (IH4)

Improvement Heuristics Algorithm #4 (IH4) was implemented as described in [2].

# 4 Neighbourhood-based Metaheuristics

## 4.1 Neighbourhood-based Metaheuristics Algorithms

The goal of the third (and last) assignment was to study, design and implement two neighbourhood-based metaheuristic algorithms. The author studied and implemented two distinct approaches: Neighbourhood-based Metaheuristics Algorithm #1 (LSH1), which results from the combination of the *simulated annealing* [3] and the *tabu search* [4] algorithms; Neighbourhood-based Metaheuristics Algorithm #2 (LSH2), which is based on the General Variable Neighbourhood Search (GVNS) algorithm [5]. The set of experiments required to fine-tune

the algorithms and the computation time of each of them only allowed to perform the complete experiments to a limited subset of the solutions obtained with the improvement heuristics algorithms (see section 5).

#### 4.1.1 Neighbourhood-based Metaheuristics Algorithm #1 (LSH1)

Neighbourhood-based Metaheuristics Algorithm #1 (LSH1) was implemented as follows:

**Algorithm 8:** LSH1 Algorithm (Simulated annealing with tabu procedure).

**Result:** LSH1 Algorithm solution.
Load current solution;
Compute current cost;
Initialise initial temperature = 10;
Initialise final temperature = 0.01;
Initialise (cooling ratio) $\alpha = 0.99$;
Initialise tabu factor = 10;
Initialise patience (threshold) = 30;
Initialise current iteration = 1;
Initialise best solution as a copy of the current solution;
Initialise best cost as a copy of the current cost;
Initialise current patience = 0;
Initialise current temperature as a copy of initial temperature;
**while** (current temperature > final temperature) *and* (current patience < patience) **do**
  Randomly select a subset of valid (*i.e.*, that maintain the universality of the solution) neighbours of the current solution (number of neighbours = 3× the number of total sets of the problem instance);
  Choose the best neighbour (*i.e.*, the neighbour with lower cost);
  Compute the cost difference = current cost − cost of the best neighbour;
  **if** cost difference > 0 **then** current solution = the best neighbour;
  current cost = the cost of the best neighbour ;
  **else**
    **if** $random[0,1) < e^{\frac{\text{cost difference}}{\text{current temperature}}}$ **then** current solution = the best neighbour;
    current cost = the cost of the best neighbour;
  **end**
  **if** current cost < best cost **then** best cost = current cost;
  best solution = current solution;
  current patience = 0;
  **else** current patience + = 1;
  Update current temperature × = $\alpha$;
  Update the tabu factor of the sets that are not available in the current solution (the algorithm will decide how to choose the neighbours based on this factor);
  Update iteration + = 1;
**end**

#### 4.1.2 Neighbourhood-based Metaheuristics Algorithm #2 (LSH2)

Neighbourhood-based Metaheuristics Algorithm #2 (LSH2) was implemented as follows:

**Algorithm 9:** LSH2 Algorithm (based on the GVNS approach).

**Result:** LSH2 Algorithm solution.
Load current solution;
Compute current cost;
Initialise $k = 0$;
Initialise $k_{max} = 10$;
Initialise $l_{max} = 10$;
Initialise current iteration = 1;
Initialise best solution as a copy of the current solution;
Initialise best cost as a copy of the current cost;
**while** $k < k_{max}$ **do**
  Randomly select a subset of valid (*i.e., that maintain the universality of the solution*) neighbours of the $N_k$ neighbourhood the current solution (number of neighbours = 3× the number of total sets of the problem instance);
  Randomly choose a $N_k$ neighbour;
  Initialise $l = 0$;
  **while** $l < l_{max}$ **do**
    Randomly select a subset of valid (*i.e.*, that maintain the universality of the solution) neighbours of the $N_l$ neighbourhood the current solution (number of neighbours = 3× the number of total sets of the problem instance);
    Choose the best $N_l$ neighbour (*i.e.*, the neighbour with lower cost);
    **if** cost of the $N_l$ neighbour < cost of the $N_k$ neighbour **then** $N_k$ neighbour = $N_l$ neighbour;
    $N_k$ neighbour cost = $N_l$ neighbour cost;
    $l = 0$;
    **else** $l+ = 1$;
  **end**
  **if** cost of the $N_l$ neighbour < cost of the current solution **then**
    current solution = $N_l$ neighbour;
    current cost = $N_l$ neighbour cost;
    **if** current cost < best cost **then** best solution = current solution;
    best cost = current cost;
  **end**
  **else** $k+ = 1$;
**end**

# 5 Results and Discussion

The following subsections present the results for the constructive and improvement heuristics, and the local-based search metaheuristics assignments of this project.

## 5.1 Constructive Heuristics

Table 1 presents the results obtained for the constructive heuristics algorithms. CH1 attains the best results regarding the average percentage deviation from best-known solutions (for both processed and non-processed solutions). CH2 and CH2, clearly benefit from the set the redundancy elimination; this can be explained by the high average percentage deviation from best-known solutions (for both processed and non-processed solutions), which suggests that these two constructive heuristics are far from optimal, thus having a clear margin for improvement on this type of post-processing. On the other hand, an attentive look at the raw results spreadsheets shows that CH5 can achieve optimal results for several individual instances, but performs very low on others, thus explaining the high values of average percentage deviation from best-known solutions (for both processed and non-processed solutions).

Table 1: Results obtained for the three constructive heuristics algorithms presented in this report. The second and third columns show the average percentage deviation from best-known solutions for both no-processed and processed solutions and the fourth column presents the fraction of instances that profit from set the redundancy elimination. Values considered relevant for the analysis are highlighted in bold.

| Algorithm | % Deviation from Optimal (No-Processing) | % Deviation from Optimal (Processing) | Fraction that Benefits from Processing |
|---|---|---|---|
| CH1 | **19.23** | **19.23** | 0.02 |
| CH2 | 460.72 | 366.48 | **1.00** |
| CH5 | 1323.85 | 454.16 | 0.76 |

## 5.2 Improvement Heuristics

Table 2 presents the results obtained for the improvement heuristics algorithms applied on both processed and non-processed solutions. For the IH1, IH2 and IH4 the best results are obtained starting from a processed solution (see values highlighted in bold); this may suggest that starting with a processed solution may be beneficial for a local-search phase. On the other hand, the best result, presented in IH3, was obtained with a non-processed solution. In fact, it may be easier for a local-search algorithm to start with a less-processed solution, allowing it to have a broader neighbourhood of solutions to evaluate, and not getting stuck too early in a local minimum in opposition to what happens when the algorithm starts with a processed-solution. This also explains the fraction of instances that profit from the local-search phase, which increases with the degree of processing of the input solution and, obviously, the execution times, which decrease with degree of processing of the input solution, which suggests that the algorithm gets stuck in a local minimum too early. Table 3 presents the results obtained for the Student t-test (one-tail, $\alpha = 0.05$) to assess the statistical difference of the four improvement heuristics algorithms applied to the different input solutions (processed or non-processed). It is possible to observe that, for this assignment, both FI and BI present no statistical difference, while the hybrid approaches seem to generate solutions with statistical difference. This can be explained by the dimensions of our instances which are relatively low when compared with problems with higher dimensions; therefore, searching for the first or for the best neighbour may be the same thing, assuming that the algorithm has a high number of iterations to perform the search.

Table 2: Results obtained for the four improvement heuristics algorithms presented in this report. The third column shows the average percentage deviation from best known solutions, the fourth column presents the total execution time (in seconds) for all the instances and the fifth column approaches the fraction of instances that profit from the local-search phase. Values considered relevant for the analysis are highlighted in bold.

| Algorithm | Applied On | % Deviation from Optimal | Total Execution Time (s) | Fraction that Benefits from Local Search |
|---|---|---|---|---|
| **IH1** | **CH1 (No Processing)** | 14.76 | **2686.17** | **1.00** |
| | **CH1 (Processing)** | **14.62** | 2709.86 | **0.98** |
| | **CH2 (No Processing)** | 80.65 | 3977.50 | **1.00** |
| | **CH2 (Processing)** | 83.88 | 4035.42 | **1.00** |
| | **CH5 (No Processing)** | 162.16 | 3849.03 | 0.86 |
| | **CH5 (Processing)** | 176.50 | 3205.92 | 0.86 |
| **IH2** | **CH1 (No Processing)** | 14.71 | 3079.54 | **1.00** |
| | **CH1 (Processing)** | **14.62** | **3006.61** | 0.98 |
| | **CH2 (No Processing)** | 80.43 | 4365.31 | **1.00** |
| | **CH2 (Processing)** | 84.12 | 4505.48 | **1.00** |
| | **CH5 (No Processing)** | 162.27 | 4062.32 | 0.86 |
| | **CH5 (Processing)** | 176.58 | 3842.74 | 0.86 |
| **IH3** | **CH1 (No Processing)** | 14.14 | 2433.12 | 1.00 |
| | **CH1 (Processing)** | 14.14 | 2664.58 | 1.00 |
| | **CH2 (No Processing)** | **8.80** | 2566.34 | **1.00** |
| | **CH2 (Processing)** | 21.89 | 2099.09 | **1.00** |
| | **CH5 (No Processing)** | 13.55 | 2155.90 | 0.86 |
| | **CH5 (Processing)** | 92.67 | **1818.08** | 0.86 |
| **IH4** | **CH1 (No Processing)** | 14.53 | 2383.58 | 1.00 |
| | **CH1 (Processing)** | **14.40** | 2360.37 | **1.00** |
| | **CH2 (No Processing)** | 80.01 | 2124.77 | **1.00** |
| | **CH2 (Processing)** | 83.14 | 1841.40 | **1.00** |
| | **CH5 (No Processing)** | 162.53 | 2055.18 | 0.86 |
| | **CH5 (Processing)** | 175.83 | **1811.86** | 0.86 |

Table 3: Results obtained for the Student t-test concerning the significant difference between the solutions generated by the different improvement heuristics algorithms.

| **Algorithms** | **Solutions Difference is Statistically Significant** | | | | | |
|---|---|---|---|---|---|---|
| | **IH1 vs IH2** | **IH1 vs IH3** | **IH1 vs IH4** | **IH2 vs IH3** | **IH2 vs IH4** | **IH3 vs IH4** |
| **CH1 (No Processing)** | No | Yes | Yes | Yes | Yes | Yes |
| **CH1 (Processing)** | No | Yes | Yes | Yes | Yes | Yes |
| **CH2 (No Processing)** | No | Yes | No | Yes | No | Yes |
| **CH2 (Processing)** | No | Yes | No | Yes | No | Yes |
| **CH5 (No Processing)** | No | Yes | No | Yes | No | Yes |
| **CH5 (Processing)** | No | Yes | Yes | Yes | Yes | Yes |

## 5.3 Neighbourhood-based Metaheuristics

Table 4 presents the results obtained for the local-based search metaheuristics algorithms applied to both processed and non-processed solutions. Due to computational processing time and deadline constraints, results for all the complete set of solutions originated by the improvement heuristics phase could not be included in this report (at the delivery moment) and will be available in the public repository created for this project (see section 1). Regarding the LSH1 algorithm (simulated annealing with tabu search), it is worth to mention that:

1. The temperature was the most difficult parameter to tune. The strategy consisted of a *trial-error* approach in which the starting point was an extremely high temperature. The author verified that for extremely high temperatures, the algorithm was generating solutions with a higher cost than the input solution. Therefore, several starting temperatures were tested until the value of 10 was reached. During this iterative process, the cooling ratio, $\alpha = 0.99$, was kept constant; in fact, this parameter never changed during the experiments.

2. The number of selected (or generated) neighbours in each iteration is very important. In addition to the temperature parameter, the author acknowledged that a higher number of available neighbours may increase the probability of escaping the current local optimum and finding a better one. Although this makes sense (*i.e.*, by increasing the number of possibilities, we are also increasing the number of possible "paths"), the author only decided to perform this test in this phase (*e.g.*, in the improvement heuristics phase, the algorithms were generating only one valid neighbour per iteration). In cooperation with the temperature, the author decided to establish, per iteration, that the number of generated neighbours would be equal to three times the number of sets of the problem instance. The throwback of this option relies on the fact that the computational time needed to run over all the sets of solutions would significantly increase. The author decided, however, that it was worth to take the risk.

3. To overcome the issues raised by the parameters presented above, the author introduced a patience variable. Widely used in machine learning [6], this trick allows to early-stop the algorithm when it could not escape the local optimum for a specific number of iterations. The value of 30 was chosen empirically, and allowed to speed-up the entire process (although it was not possible to obtain the results for every set of solutions originated in the improvement heuristics phase).

Regarding the LSH2 (GVNS), the parameters that required attention were the $k_{max}$ and $l_{max}$. The number of selected neighbours in each iteration was the same as in LSH1; therefore, the *trade-of*, in this case, is related to the number of cycles that the algorithm can achieve if it is always capable of escaping local optima. The approach to choose the parameters, in this case, was, somehow, naive. The intuition was based on the fact that one should assure that the algorithm had enough iterations to promote the finding of a better solution, thus, escaping local optima. The author acknowledges that the values of $k_{max} = 10$ and $l_{max} = 10$ are too high and jeopardise the computational time. For this reason, it was not possible to report any results for the LSH2 algorithm at the report delivery moment. Using the Student t-test (one-tail, $\alpha = 0.05$), the author concludes that there is no evidence of the statistical difference between both of the reported results.

Table 4: Results obtained for the LSH1 algorithm presented in this report. The first column presents the type of solution that was given as input (*e.g.*, "IH1-CH1 w/out processing" means that the solution that was given was the result of the IH1 algorithm on the CH1 solution without set redundancy removal), the third column shows the average percentage deviation from best-known solutions, the fourth column presents the total execution time (in seconds) for all the instances and the fifth column approaches the fraction of instances that profit from this neighbourhood-based phase. Values considered relevant for the analysis are highlighted in bold.

| Algorithm | Applied On | % Deviation from Optimal | Total Execution Time (s) | Fraction that Benefits from Local Search |
|---|---|---|---|---|
| **LSH1** | **IH1-CH1 w/out processing** | 11.82 | **162621.50** | **0.88** |
| | **IH1-CH1 w/ processing** | **11.67** | 169528.28 | 0.83 |

# 6 Conclusions

This report presents an exploratory study of constructive and improvement heuristics algorithms, and neighbourhood-based metaheuristics for the conclusion of a three-phase assignment project under the scope of the curricular unit "Heuristics and Metaheuristics". Results show that the quality of the solutions improves through the three phases, however, with the trade-of of both processing times and costs. Further work should be devoted to the study of new strategies to increase the quality of the solutions obtained up to similar optimum values reported in literature [1, 2]. Possible lines of work include the study of different constructive or improvement heuristics algorithms, the study of the impact of their parameters, the study of the generation of neighbour solutions.

# References


[1] J. E. Beasley, "An algorithm for set covering problem," *European Journal of Operational Research*, vol. 31, no. 1, pp. 85–93, 1987.

[2] F. Akhter, "A heuristic approach for minimum set cover problem," *Int. J. Adv. Res. Artif. Intell*, vol. 4, pp. 40–45, 2015.

[3] D. Delahaye, S. Chaimatanan, and M. Mongeau, "Simulated annealing: From basics to applications," in *Handbook of Metaheuristics*, pp. 1–35, Springer, 2019.

[4] M. Gendreau, "An introduction to tabu search," in *Handbook of metaheuristics*, pp. 37–54, Springer, 2003.

[5] P. Hansen and N. Mladenović, "Variable neighborhood search: Principles and applications," *European journal of operational research*, vol. 130, no. 3, pp. 449–467, 2001.

[6] L. Prechelt, "Early stopping - but when?," 03 2000.