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

Heuristics for the Set Covering Problem

Tiago Filipe Sousa Gonçalves

October 21, 2020

Contents

1 Introduction

Under the scope of the curricular unit "Heuristics and Metaheuristics", the students were proposed a two-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) and the second phase consisted on the study and proposal of different improvement heuristics (see section 3). 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
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, α 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
    end
    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
    end
    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 ∈ [0, n);
Start empty solution;
while number of rows already covered < number of rows to be covered do
    for each x*_j > 1/f do
        add x*_j to the solution
    end
    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 this 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;
        end
        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
    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 = current iteration + 1;
        current patience value = 0;
    end
    else
        current iteration = current iteration + 1;
        current patience value = current patience value + 1;
    end
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;
        end
        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
    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 = current iteration + 1;
        current patience value = 0;
    end
    else
        current iteration = current iteration + 1;
        current patience value = current patience value + 1;
    end
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;
            end
            else
                if all the sets are in the chosen sets list then
                    reboot chosen sets list as an empty list;
                end
                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
        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;
        end
        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
    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 = current iteration + 1;
        current patience value = 0;
    end
    else
        current iteration = current iteration + 1;
        current patience value = current patience value + 1;
    end
end
```

3.1.4 Improvement Heuristics Algorithm #4 (IH4)

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

4 Results and Discussion

The following subsections present the results for the both constructive and improvement heuristics parts of this assignment.

4.1 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

4.2 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 Conclusions

This report presents an exploratory study of constructive and improvement heuristics algorithms for the conclusion of a two-phase assignment project under the scope of the curricular unit "Heuristics and Metaheuristics". Further work should be devoted on the study of new strategies to increase the quality of the solutions obtained for both constructive and improvement heuristics algorithms (*e.g.*, different constraints, different strategies for the neighbour-search phase).

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.