U.Porto Journal of Engineering, 5:1 (2019) 1-15
ISSN: 2183-6493
DOI: 10.24840/2183-6493_005.001_0001

Received: 4 December, 2018
Revised: 27 December, 2018
Published: 26 April, 2019

Two Neighbourhood-based Approaches for the Set Covering Problem

Eduardo Castro[1]
[1]INESC Technology and Science - Associate Laboratory – INESC TEC, Porto, Portugal; Doctoral Program in Electrical and Computer Engineering, Faculty of Engineering, University of Porto, Porto, Portugal (emcastro@inesctec.pt) ORCID (0000-0003-4144-695X)

Abstract
The Set Covering Problem is a well-known NP-complete problem which we address in this work. Due to its combinatorial nature heuristic methods, namely neighbourhood-based meta-heuristics, were used.
Based on the well-known algorithms GRASP, Simulated Annealing and Variable Neighbourhood Descend, along with a constructive heuristic based on a dynamic dispatching rule to generate initial feasible solutions, two approaches to the problem were formulated. The performance of both methods was assessed in 42 instances of the problem. Our best approach has an average deviation from the best-known solution of 0.23% and reached 0% for 26 instances under 40 minutes.

Author Keywords. Neighbourhood-based Metaheuristics, Metaheuristics, Set Covering Problem, Optimization, Variable Neighbourhood Descent, Simulated Annealing, Greedy Randomized Adaptative Search Procedure.

Type: Research Article
Open Access Peer Reviewed CC BY

1. Introduction

The Set Covering Problem (SCP) is a classic computer science problem shown to be NP-complete in 1972 (Karp 1972). Originally, the problem was defined as: *"given a finite family of sets and a positive integer k, is there a subfamily of sets with cardinality smaller than or equal to k and whose union is the same as the original family?"*.

The demonstration that this problem is NP-Complete ensures that, by computational complexity theory, there are no deterministic polynomial time algorithms that solve all instances of the problem unless the widely believed conjecture P≠NP is proven to be false (Fortnow 2009).

When addressing this kind of problems some authors resort to exact algorithms. These focus on finding the optimal solution to the SCP problem, usually relying on branch-and-bound (Balas and Carrera 1996). However, due to this problem being NP-hard, the time taken by these algorithms can grow exponentially as the size of the problem increases, particularly in worst case scenarios. An alternative approach is to use approximate methods which find non-optimal solutions but provide some guarantee of how close the solution is to optimality. Gomes et al. (2006) compared multiple approximation algorithms both in terms of their theoretical and empirical performance. Finally, there are heuristic algorithms which find solutions to the problem which are neither optimal nor present any guarantee of optimality. However, they run in acceptable time often with fairly good results (Beasley and Chu 1996; Yelbay, Birbil, and Bülbül 2014). In this work we resort to this last alternative.

A variant of the SCP (Beasley 1987) was tackled. In this problem, each set has an associated cost. The objective is to minimize the sum of the costs associated with all sets in the selected subfamily. It is easy to see that this is a generalization of the original SCP, as shown in section 3. The formulation considered is not just of theoretical importance. In many different scenarios this problem can be of great practical use. Consider for instance the problem of setting up a facility for a certain activity. A set of tools need to be bought to perform several different specific tasks. Each tool can be used for a finite set of tasks. The problem of selecting which tools to buy so that all tasks can be performed while minimizing the cost is equivalent to the formulation hereby presented. Although this is just a potential example, concrete practical applications are known for the problem in areas like airline crew scheduling (Baker et al. 1979), location of emergency facilities (Revelle, Marks, and Liebman 1970), or routing problems (Foster and Ryan 1976).

To address the SCP, we propose two neighbourhood-based metaheuristic algorithms. Both approaches share the same initial phase, where a constructive heuristic is used to obtain an initial feasible solution. With this starting point, both algorithms do local search considering a neighbourhood of feasible solutions. The approaches differ in the considered neighbourhood and in the process used to escape local optima. While the first uses a swap operation to generate new neighbours and Variable Neighbourhood Descend, the second uses set addition and removal with Simulated Annealing.

The rest of the paper is organized as follows: section 2 frames the work in the context of previous research for the SCP problem, section 3 presents the problem in a formal way along with a detailed description of the developed methods. In section 4 we present our experiments and the obtained results. Finally, we present our main conclusions in the last section.

2. Related Work

Heuristics are non-optimal methods designed for a specific problem. While they do not guarantee optimality, they often provide satisfactory solutions. Metaheuristics are more general in the sense that they can be applied to many optimization problems. Due to this, two approaches to one problem can be different even if they use the same metaheuristic (Talbi 2009, xvii).

An important class of heuristic methods in the context of this work are constructive heuristics. These are greedy algorithms which start with an empty solution and iteratively assign values to the decision variables until a complete solution is generated. Although we do not aim at an exhaustive description of these methods, a review of these methods for the SCP problem was done by Crawford et al. (2018).

Metaheuristics work by iteratively evaluating solutions in the search space. Importantly, they only search a very small subset of the search space, otherwise the amount of time required to run the algorithms would make them impractical.

Metaheuristics can be divided in two groups: 1. Population-based and 2. neighbourhood-based. While population based keep a population of solutions which continually explore the search space, neighbourhood-based have a single solution which is slightly adjusted at each iteration. There are several population-based metaheuristics that have been applied to the SCP problem, such as genetic algorithms (Aickelin 2002) and ant-colony optimization (Ren et al. 2010).

In this work we focus on neighbourhood-based approaches based on three metaheuristics: 1. greedy randomized adaptative search procedure (GRASP) (Feo and Resende 1989); 2. Simulated Annealing (SA) (Kirkpatrick, Gelatt, and Vecchi 1983) and 3. Variable Neighbourhood Descend (VND) (Hansen, Mladenović, and Moreno Pérez 2010). Although there are some approaches in the literature that make use the same metaheuristics to address the same problem our proposed approaches are original and thus, have never been evaluated.

3. Methods

3.1. Formal description of the problem

In this work the following SCP variant was considered:

Given:

A set of elements, U(the universe)

A collection of subsets of U, S

A cost $c_i$associated with each element of S_i

Find the sub-collection of S, V that minimizes:

$\sum_{S_i \in V} c_i$

Subject to:

$\cup V = U$

Note that, if we set the cost of all sets to 1, we have the original SCP. In this formulation a solution for the problem is given by V, we say it is feasible if $\cup V = U$ and the objective function we want to minimize is $\sum_{S_i \in V} c_i$. The notation used in the rest of the paper is consistent with the problem formulation stated above.

3.2. Constructive heuristics

The first step in both proposed approaches is to generate an initial feasible solution, with constructive heuristics, which will later be refined. The general idea of constructive heuristics is to start with an unfeasible solution and iteratively approximate it to a complete one. In this work V is empty at the start, and, in each iteration, a single subset is added to our solution. The subset is chosen as the one that maximizes a certain criterion, which is evaluated in each iteration. This process is usually known as a dynamic dispatching rule.

Two criteria were considered are:

- Elements per Cost ratio (EC) - Ratio between new elements covered and the cost of the set.
- Normalized Elements per Cost ratio (NEC) – Similar to EC but the new elements covered are weighted by the cost of the cheapest set containing that element.

In Figures 1, 2 and 3 the pseudo code for the constructive heuristic and for both criteria is shown. Note that the complexity of both these algorithms is $O(n_{elems})$ as in each iteration one set that covers at least one element is added to the solution. In practice the cardinality of the final solution is often smaller than the number of elements to cover. Also, in each step, the NEC criterion is slightly more expensive due to requiring an additional operation.

In this work a non-greedy version of the algorithm is also defined by introducing parameter α. In this version the selected subset is taken randomly from the α subsets with the highest value for the criterion.

```
Algorithm 1 Constructive Heuristic for SCP
  Input: $S, C$
  Output: $V$
 $V \leftarrow \{\}$
 while $\bigcup V \neq U$ do
     $R \leftarrow rank\_subsets(S, C, V)$
     $i \leftarrow argmax(R)$
     $V \leftarrow V \cup \{S_i\}$
 end while
```

Figure 1: Pseudo-code for the dynamic dispatching rule method

```
Algorithm 2 EC Ranking Datasets
  Input: $S, C, V$
  Output: $R$
 $R \leftarrow \{\}$
 for $i \leftarrow 1$ to $|S|$ do
     $R_i \leftarrow [\sum_j 1]/Ci$ with $j \in S_i, j \notin \bigcup V$
 end for
```

Figure 2: Pseudo-code for the Elements per Cost criterion

```
Algorithm 3 NEC Ranking Datasets
  Input: $S, C, V$
  Output: $R$
 $R \leftarrow \{\}$
 $D_j \leftarrow min(C_i)$ s. t. $j \in S_i$
 for $i \leftarrow 1$ to $|S|$ do
     $R_i \leftarrow [\sum_j 1 \times D_j]/Ci$ with $j \in S_i, j \notin \bigcup V$
 end for
```

Figure 3: Pseudo-code for the Normalized Elements per Cost criterion

3.3. Neighbourhood-based metaheuristics

Neighbourhood-based approaches slightly change the current solution iteratively. The concept of neighbourhood can vary, but it consists on solutions close to the one being considered. The number of neighbours should be very small when compared to the number of feasible solutions in the search space. Otherwise finding neighbours would be a combinatorial problem on its own.

Local search consists in, first, evaluating neighbours and then, moving the solution to a better neighbour. Local search can be done using a *best-N* approach where the solution is moved to the best neighbour or using a *first-N* where the solution is moved to the first neighbour that is better than the current solution. One can also consider intermediate approaches. While the *best-N* strategy finds better solutions in each iteration, the *first-N* introduces randomness in the algorithm while taking less time per iteration.

By always selecting a better neighbour, the local search algorithm inevitably gets trapped in local optima, when all neighbours are worse than the current solution. This does not mean there are no better solutions in the search space, only that if there are, they are not in the considered neighbourhood structure. To avoid this, different methods can be considered which allow the solution to escape from an "all-worse" neighbourhood structure. In this work three were considered: GRASP, SA and VND.

3.4. Variable neighbourhood descent approach

One way to escape local optima is by expanding the neighbourhood structure in hope of finding a better neighbour there. Variable Neighbourhood Descent (VND) is a variant of a bigger group of methods, Variable Neighbourhood Search, in which instead of one, a sequence of neighbouring structures is defined. In short, during VND, local search is performed in a neighbourhood structure. If a better solution is found the process starts again with that solution as the initial point and considering the first neighbourhood structure. Otherwise, the neighbourhood is expanded.

3.4.1. Neighbourhood structures

In the context of this work the first neighbourhood structure (N1)was defined as all possible V's which can be obtained by swapping one subset in the current solution by one or zero subsets not included in the current solution. The second neighbourhood structure (N2) is defined analogously, by swapping one or two subsets of the current solution by zero, one or two non-selected subsets. Notice that N2 contains N1. This is not a good practice since if N2 is reached, N1 has already been searched without yielding a better solution. However, given the fact that the number of neighbours in N1 is usually very small compared to the number of neighbours in N2, the loss in performance is negligible, while allowing a simpler implementation. Further, more neighbourhood structures could be defined analogously, but they are unfeasible in practice, as the number of neighbours increases exponentially as the maximum number of sets in the swap increases. Notice that the inclusion of the empty set in the formulation of N1 and N2 (the operation allows a swap of x sets by zero sets), ensures that the number of selected subsets is not invariant in the process and, more importantly, that redundancy elimination is done implicitly.

3.4.2. Reducing the number of candidates

Given the number of neighbours the algorithm evaluates for each run, a good selection of which subsets should be considered in N1 and N2 can drastically decrease the complexity of the algorithm. We specify here how neighbours were generated in such a way that the number of unfeasible or more expensive solutions generated and evaluated is minimized.

Considering C_{out} as the list of possible removals. In this list, each element can contain one or two sets, depending whether N1 or N2 is being considered. For N1, C_{out} is equal to V. Any set can be taken out. For N2 the Cartesian-product between C_{out} and itself is performed. This product yields elements which contain the same set duplicated. When this is the case, one of these elements is removed. It also returns two solutions for each combination (e.g. (S_1, S_2) and (S_2, S_1) would both be in C_{out}). One of these is also removed. Members in C_{out} are ordered by the number of elements that will become uncovered if that member is selected. This way solutions where the sets removed lead to a feasible or close to feasible solution are evaluated first. After selecting a candidate to be taken out, c_o, a list of candidates which can potentially be added to the set, C_{in}, is generated. This is done also using the Cartesian-product but including the empty set. Subsets not present in V, which cover at least one element to be left uncovered by the removal of c_o, and which have an associated cost smaller than the sum of costs of all subsets in c_o are considered.

By removing redundancy in the generated C_{out}, ordering its elements according to how close the generated solution will be to feasibility, and generating C_{in} with cheaper and relevant (which cover potentially uncovered elements) subsets, the number of evaluated solutions is drastically reduced, making the algorithm significantly more efficient. This is vital as it allows mixing VND with GRASP as shown later.

3.4.3. Best-N vs First-N

As seen before the number of neighbours in N2 is exponentially bigger than in N1. Due to this a First-N approach was selected for N2. For N1 Best-N local search was selected. As shown in the next subsection for each instance of the problem, we propose to run this algorithm multiple times with different initial solutions. We consider that this source of diversification is enough to guaranty a good exploration of the search space, potentially decreasing the benefits of a First-N strategy in N1.

3.4.4. GRASP and VND

Greedy Randomized Adaptive Search Procedure (GRASP) is a well-known method to increase the diversification of a search algorithm. Typically, in neighbourhood-based approaches, a feasible solution is constructed and optimized with local search. One way to escape a local minimum is to run local search again using a different starting point. This is the logic behind GRASP: first, multiple feasible solutions are generated, and then, improved separately using a local search method.

If we consider the VND method as our local search approach (even though two different neighbourhoods were used), running VND on different initial solutions is becomes very similar to GRASP. Given the fact that evaluating a N3 neighbourhood is unfeasible in practice, given a time constraint, we propose to start the VND search again using a different initial solution.

Algorithm 4 Variable Neighborhood Descend with GRASP for SCP

```
  Input: $S, C$
  Output: $V$
 $t \leftarrow time()$
 while $time() - t < t_{max}$ do
   $V \leftarrow ch(S, C)$
   $N \leftarrow 1$
   while $N < 3$ do
     $C_{out} = V$
     if $N = 2$ then
       $C_{out} \leftarrow C_{out} \times C_{out}$
       $C_{out} \leftarrow remove\_duplicates(C_{out})$
     end if
     $C_{out} \leftarrow order(C_{out}, key = uncovered(V, C_{out}))$
     for $c_o$ in $C_{out}$ do
       $C_{in} \leftarrow generate\_Cin(V, c_o)$
       if $N = 2$ then
         $C_{in} \leftarrow C_{in} \times C_{in}$
         $C_{in} \leftarrow remove\_duplicates(C_{in})$
       end if
       for $c_i$ in $C_{in}$ do
         if $sum(cost[c_i]) < sum(cost[c_o])$ then
           if $valid([V \setminus c_o] \cup c_i)$ then
             $V \leftarrow [V \setminus c_o] \cup c_i$
             $N \leftarrow 1$
             Goto while
           end if
         end if
       end for
     end for
     $N \leftarrow N + 1$
   end while
 end while
```

Figure 4: Pseudo-code for the Variable Neighborhood Descend approach with GRASP

3.5. Simulated Annealing

Annealing is a well-known process in materials science. The main idea is that for some materials, a heating phase followed by a controlled cooling phase can alter the properties of the material, namely its ductility and its hardness. Applying a similar logic to the problem of SCP can allow iterative algorithms to escape local optima, by randomly jumping to a worse solution in the neighbourhood structure. The chance of this happening depends on how worse the new solution is compared to the current one and the current value of parameter T (of temperature). T usually decays exponentially. Due to the similarities with the physical process, this technique is called Simulated Annealing. In the version hereby described, the algorithm has a probability of moving of less than one when it finds a better solution, which is not the standard formulation.

3.5.1. Neighbourhood structure

In this method the neighbourhood is generated by either adding or removing one set from V. Essentially, a random set is selected, S_i, and if $S_i \in V$, the new solution is obtained by removing it, assuming the result is feasible. If $S_i \notin V$ then the new solution is obtained by adding S_i to V. There are two reasons which can lead the algorithm to reject the generated solution: i) removing a set leads to an unfeasible solution; ii) based on the probabilistic approach the event does not occur. When this happens, the algorithm selects another random set. Pseudo-code for the proposed algorithm is shown in Figure 5.

3.5.2. Chance of selecting a solution

Given a current solution v_c and a feasible neighbour v_n the probability of the algorithm setting v_c to v_n is given by Equation 1 if the solution is obtained by addition and Equation 2 if the solution is obtained by removal.

$$P(V \leftarrow V \cup \{S_i\}) = e^{-C_i/T} \tag{1}$$

$$P(V \leftarrow V \setminus \{S_i\}) = \left(1 - e^{-C_i/T}\right) \tag{2}$$

3.5.3. Selecting values for T and α

Optimization requires two parameters: i) an initial value for T and ii) the decay rate α.

These parameters can greatly affect the cost of the final solution and there are no values that work for all instances of the problem. Optimal parameters might depend on the data, the neighbourhood structure or the objective function. In this subsection we present a heuristic method to determine good values for these variables depending on the considered instance.

First an initial value for T is fixed, T_{max}. It is easy to see that if this value is too high the objective function will greatly increase in the first iterations. If it is too low, the objective function will decrease rapidly. Both these phenomena are undesirable as they can lead to worse final solutions, when compared to the ones that could be obtained with a more careful parameter initialization. Additionally, for efficiency, we want to spend as much time as possible with a good value of T, during optimization. We propose to select T_{max} so that the expected value of the objective function of the first iteration is equal to the starting solution. By noticing that in the first iteration either a set is added or removed, this expected value is given by Equation 3. Let v_0 and v_1 be the values of the objective funtion for the initial and after-first iteration solutions. Then:

$$E[v_0 - v_1] = \frac{1}{|V|}\left[\sum_{S_i \notin V} e^{-C_i/T} \times C_i - \sum_{S_i \in V} \left(1 - e^{-C_i/T}\right) \times C_i\right] \tag{3}$$

To obtain T_{max} this function is evaluated in N equally spaced points in the logarithmic space (assuming the function crosses zero in this range) and the point with a smaller absolute value is selected. Notice that in the above expression it is assumed that unfeasible solutions can be generated. In practise sets which its removal would lead to unfeasible solutions are not counted in the mean. Then a value for T_{min}, the temperature in the last iteration, is selected such that the probability of adding/removing the cheapest set is equal to 5%. We argue that this is a good point to end because a better solution is being selected 19 out of each 20 feasible neighbours, which is very close to simple local search. The computation of α is done based on the selected number of iterations between T_{max} and T_{min}. Using this method, the only parameter that needs to be set is the number of updates. Notice that an initial solution with no redundancy would lead to a very low T_{max}, as the expected value for the variation of the objective function would be positive. As such, this heuristic method is not suited for initial solutions which are already locally optimized.

```
Algorithm 5 Simmulated Annealing for SCP
  Input: $S, C, V$
  Output: $V$
  $T = T_{max}$
  while $T > T_{min}$ do
    candidates $\leftarrow$ rand_sort($S$)
    for $S_i$ in candidates do
      if $S_i \in V$ then
        if rand()> $e^{-C_i/T}$ and valid($V \setminus \{S_i\}$) then
          $V \leftarrow V \setminus \{S_i\}$
        end if
      else
        if rand()< $e^{-C_i/T}$ then
          $V \leftarrow V \cup \{S_i\}$
        end if
      end if
    end for
    $T \leftarrow T \times \alpha$
  end while
```

Figure 5: Pseudo-code for the Simmulated Annealing stategy

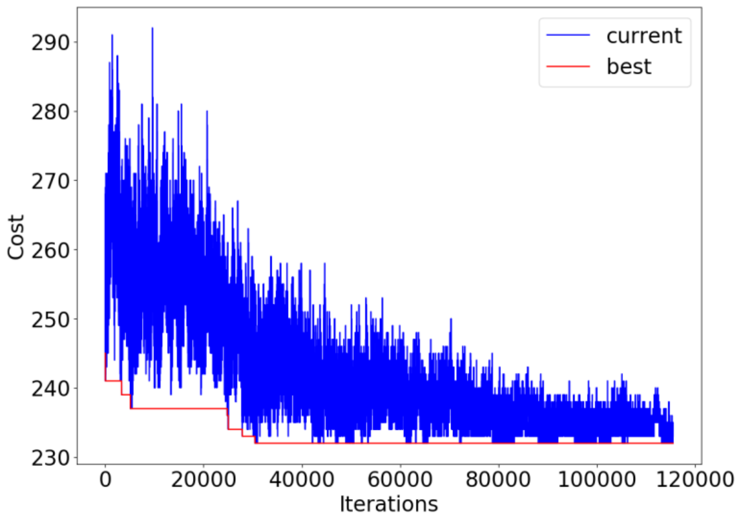

Figure 6: Objective function for the simulated annealing method using the proposed heuristic to find and initial T_{max} and α. As shown in the image, initially the objective function oscilates around the same initial value (260) instead of a quick rise or fall

4. Experimental Work

4.1. Instances of the problem

In total 42 instances of the SCP were considered in order to evaluate the performance of the proposed methods. These instances have previously been used by other authors (Beasley 1987) and are available online (Beasley, n.d.). They can be divided in seven groups of problems with similar statistics. The statistics for each group are depicted in Table 1.

				Set Cardinality			Element Minimal Cost		
Ins type	**Opt.cost**	**N_sets**	**N_elems**	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**
4	519.63	1000	200	1.00	3.99	10.63	1.00	5.49	35.00
5	256.33	2000	200	1.00	3.98	11.00	1.00	2.83	14.22
6	144.20	1000	200	2.20	9.91	19.00	1.00	2.43	12.60
A	241.40	3000	300	1.00	6.03	16.80	1.00	2.23	10.80
B	75.20	3000	300	4.00	14.96	29.00	1.00	1.31	4.40
C	224.60	4000	400	1.00	7.99	19.80	1.00	1.82	9.60
d	64.20	4000	400	7.20	20.03	37.00	1.00	1.11	3.60

Table 1: Statistics for each group of instances. The values are averages across all instances of each type. Set cardinality is the number of elements in a subset. Element minimal cost is the cost of the cheapest subset containing that element

All experiments were run on a Quad-Core Intel® Core™ i7-6700K CPU @ 4.00GHz computer with 64 GB of RAM memory. Only one CPU core was used in all experiments.

4.2. Construction phase

To generate initial feasible solutions, the proposed constructive heuristic algorithm using the EC and NEC criteria was used. A comparison per instance type was made between the two, while fixing the parameter α as 1. In this comparison the same behaviour, shown on Figure 7 (left), was observed in almost all instances. The NEC criteria seems to be a less greedy approach as it considers more expensive elements first, with a faster cost increase in initial iterations, while leaving cheaper elements to be covered later. For most instances this criterion lead to cheaper solutions with a smaller number of subsets, which reduces the number of iterations needed to find a feasible solution. This reduction compensates for the added computational cost of weighting the elements, as both algorithms took around the same time to find solutions for all instances.

In all 42 instances only two solutions had a higher cost with the NEC criteria and only one had higher cardinality. The benefits of using NEC criteria for each instance type are shown in Table 2.

As shown later, in some cases it is beneficial to use $\alpha > 1$ to increase how diverse the solutions found by the constructive heuristic are. In Figure 7 (right) the distribution of the cost for 100 solutions obtained using the NEC criteria for different values of α is shown. As expected, as α increases, the distribution becomes more disperse and the mean solution has a higher cost. However, by following a less-greedy approach, better solutions than the one obtained with the deterministic approach ($\alpha = 1$) can occasionally be found.

	Solution cost reduction			Solution cardinality reduction		
Inst type	**Min**	**mean**	**max**	**min**	**mean**	**max**
4	2.3%	5.3%	7.9%	4.3%	11.1%	16.5%
5	3.9%	5.7%	7.3%	8.1%	11.3%	13.9%
6	3.7%	5.7%	8.1%	4.4%	10.1%	15.2%
A	5.7%	6.8%	7.7%	10.5%	12.5%	15.9%
B	-1.2%	5.0%	9.8%	0.0%	6.4%	12.0%
C	2.6%	5.2%	9.1%	6.0%	8.5%	15.7%
D	-4.3%	3.5%	7.2%	-4.3%	4.2%	9.4%

Table 2: Solution Cost and Cardinality reduction due to the use of the NEC criterion instead of the EC

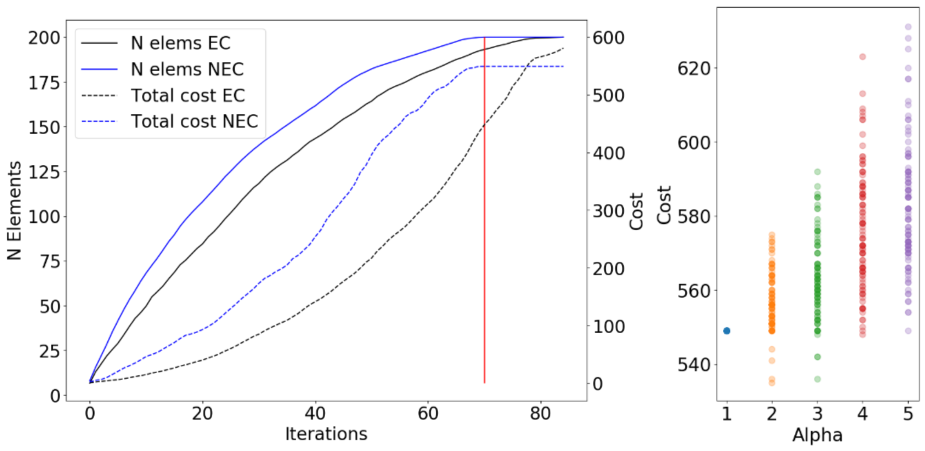

Figure 7: Left - Comparison between EC and NEC criteria on the first instance of group 4. The red line indicates the iteration in which NEC found a feasible solution while EC is still running. This behaviour was observed in almost all instances. Right - Distribution of the objective function value for 100 solutions obtained for each different value of α. If we can afford to run the algorithm multiple times a non-greedy approach is likely to yield a better final solution

4.3. Variable neighbourhood descent with GRASP

The proposed VND method requires a starting feasible solution which is obtained by the constructive heuristic method. For this the NEC criteria was used as in most cases it led to better solutions using a less-greedy approach. Selecting a value for parameter α is of major importance, as it will control how diverse our initial solutions are. This selection is highly related to the time available to perform optimization. If the available time only allows the analysis of one solution, $\alpha = 1$ should probably be selected while if infinite time is available, all possible initial solutions should be analysed.

In the scope of this work it would be unfeasible to optimize this value for all instances. Due to this a small experiment was performed to better understand the impact of α for the first instance of the problem. We run the algorithm 100 times for different values of α. Figure 8 is a 2-dimensional representation of the final solutions, obtained with Latent Semantic Analysis (Dumais 2004). In the representation, it is shown that as α increases the spreading of the final solutions also increases. In other words, they become more diverse. In this experiment, the minimum obtained cost was 514 for $\alpha \in \{1,2\}$, 512 (best known solution) for $\alpha = 3$ and 513 for $\alpha = 4$. Given these two insights, and because we are estimating to run the algorithm a few hundreds or thousands of times for each instance, $\alpha = 3$ was selected. Interestingly, performing the same experiment with 1000 solutions yielded the best-known solution for all values of α except 1, the greediest approach.

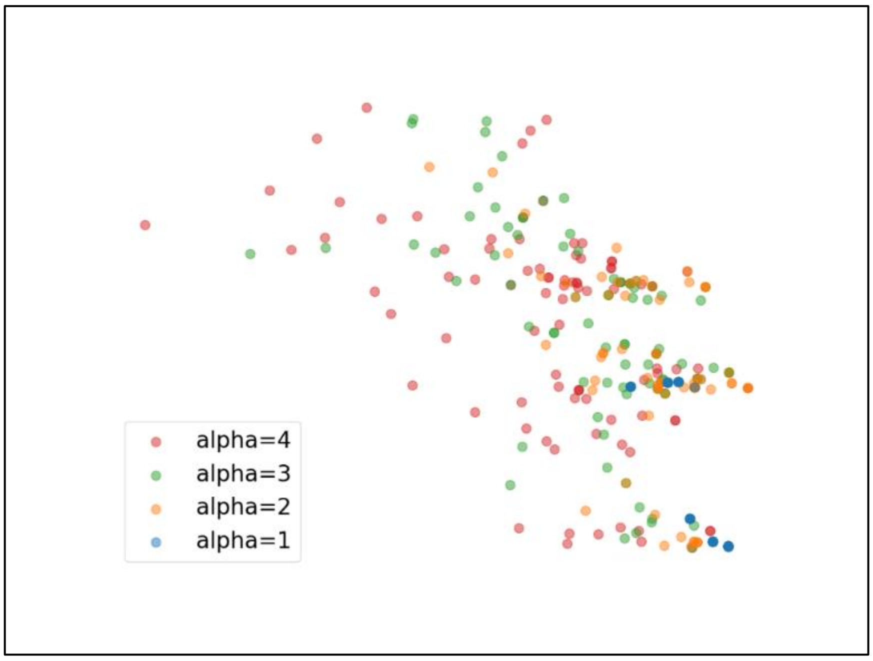

Figure 8: Distribution of feasible solutions in the search space obtained using the NEC criterion for different values of α. This representation was obtained by running the algorithm 100 times for each value of α and applying dimentionality reduction on the solutions found using Latent Semantic Analysis

For each instance the algorithm was run for one hour. Whenever VND could not find a better solution in neighbourhoods N1 and N2 a new solution was constructed and optimized. Due to the careful selection of candidates explained in subsection 3.4.2, searching N1 and N2 was very fast, with the algorithm taking less than 1 second to analyse the current neighbourhood. We tried to consider N3 but in all cases tested the neighbourhood could not be completely searched in less than an hour and it never found a better feasible solution in that time frame. This is due to the combinatorial nature of the neighbourhood structure selected. The average deviation from the best-known solutions per instance type is shown in Table 3.

Instance type	5 min.	10 min.	30 min.	60 min.
4	0.42%	0.37%	0.30%	0.28%
5	0.55%	0.35%	0.09%	0.09%
6	0.41%	0.41%	0.00%	0.00%
A	0.66%	0.66%	0.66%	0.49%
B	0.00%	0.00%	0.00%	0.00%
C	1.25%	1.16%	0.80%	0.80%
D	0.86%	0.86%	0.30%	0.00%

Table 3: Mean deviation from the best-known solution per instance type for the VND algorithm

By analysing the results, we can see that for all instances of type **6**, **b** and **d**, the proposed approach finds the best-known solution under 1 hour. These correspond to instances with subsets of high cardinality. Instance type **c** seems to be the hardest at least for this algorithm. In total 17 best-known solutions were found under 5 minutes and 26 under 1 hour. The average deviation from the best-known solution was 0.58% for a time frame of 5 minutes and 0.23% for a time frame of 60 minutes. Importantly, all best solutions were found under 40 minutes as shown in Figure 9, revealing that the benefit of running the algorithm for more than this time frame was none. A possible cause for this is that the algorithm might be repeatedly optimizing the same solutions. For the first instance, the ratio of unique to total solutions was computed for the constructive heuristic and for the VND approaches. We found only 84% were unique for the former and only 40% were unique for the latter. A method to manage optimization with memory, in order to avoid evaluating already computed solutions, could be beneficial in this case. Alternatively, a greater source of diversification could also lead to improved results.

4.4. Simulated Annealing

The proposed SA method also requires a feasible initial solution. Taking in consideration that simulated annealing is more random than the VND approach and that we expect it to run less times in a one-hour time frame, the value of α should be smaller than 3. As such we decided to perform all experiments twice, using $\alpha = 1$ and $\alpha = 2$. We called this methods SA1 and SA2.

Additionally, for the SA algorithm we need to specify the number of updates in one run of the algorithm. We selected this so that optimization takes approximately 5 minutes. Obtaining the value for the number of updates was done by trial and error on the first instance of each type. If the number of updates led to the algorithm running between 4 and 6 minutes the value was considered acceptable. Each algorithm was run 12 times (approximately 1 hour). Results are depicted in Table 4 for SA1 and Table 5 for SA2.

Both SA1 and SA2 can find all best-known solutions for instances of type **b** and **d**. For these two types, VND was also able to find all best-known solutions, suggesting these instances are less challenging than the others. As shown in Figure 9. The SA1 approach seems to be better for a single run while SA2 is superior when we run the algorithm multiple times. By means of paired T-test we determined that this difference was not statistically significant for any number of runs for a significance level of 5%. For SA1 with a time frame of approximately 5 minutes the average deviation from best known solutions was 1.97% with this value being 0% for 9 instances, while for SA2 these values being 2.17% and 4. For approximately 60 minutes this values change to 1.23% and 15 for SA1 and 0.70% and 17 for SA2.

Instance type	**5 min.**	**10 min.**	**30 min.**	**60 min.**
4	1.79%	1.27%	1.15%	1.15%
5	1.78%	1.46%	1.24%	1.16%
6	3.23%	2.94%	2.94%	2.94%
A	2.92%	2.43%	2.27%	1.93%
B	0.51%	0.51%	0.25%	0.00%
C	2.91%	2.00%	1.57%	1.57%
D	0.88%	0.56%	0.00%	0.00%

Table 4: Mean deviation from the best-known solution per instance type for the SA1 algorithm

Instance type	**5 min.**	**10 min.**	**30 min.**	**60 min.**
4	1.48%	0.99%	0.60%	0.37%
5	2.07%	1.53%	1.06%	1.06%
6	2.84%	1.68%	1.18%	1.06%
A	2.22%	1.50%	1.25%	1.25%
B	1.90%	0.52%	0.00%	0.00%
C	2.81%	1.89%	1.16%	1.06%
D	2.39%	0.58%	0.00%	0.00%

Table 5: Mean deviation from the best-known solution per instance type for the SA2 algorithm

4.5. Comparison between the two approaches

In this subsection we compare the results between both approaches one using SA and the other VND. A comparison in terms of average deviation from the best-known solution and the number of best-known solutions found can be found in Figure 9.

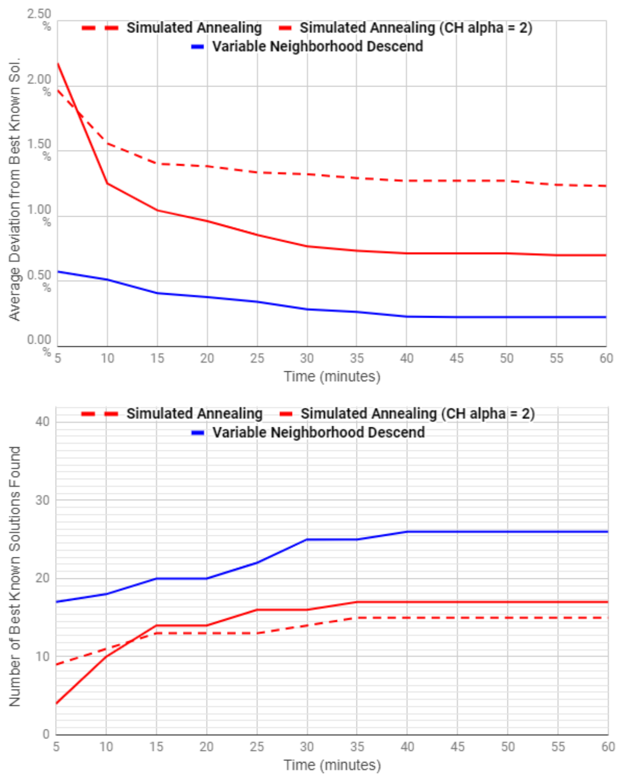

Figure 9: Performance metrics for each of the considered approaches in function of running time. Top - Average deviation from best known solution in percentage. Bottom - Number of best known solutions found, in a universe of 42 instances

The VND approach is superior in both metrics regardless of setting α to 1 or 2 in the second approach. This difference was statistically significant for all time frames considered, using a paired T-test with a significance level of 1%. The difference in performance between the two methods can be explained by multiple factors. The VND approach has a very efficient intensification phase which takes relatively little time, due to the considered neighbours having a high probability of being feasible and better than the current solution. This efficiency in generating new solutions allows us to run the algorithm many times in a short time frame for different initial solutions, similar to GRASP.

In the case of SA, we did not optimize the parameter which refers to the number of updates, we simply tried to have a 5-minute running interval. For instance, a better strategy could be to run the algorithm for 1 minute 5 times. In other words, we do not know if the cooling rate is being unnecessarily slow or too fast. A better estimation of the initial Temperature, cooling rate and number of iterations could lead us to better results. Further, the neighbourhood of the SA algorithm is different from the VND one and so, the solutions obtained might benefit from a post-processing VND search.

5. Conclusion

In this work we considered a variant of the SCP problem and designed two neighbourhood-based approaches to address it. Both approaches work on the space of feasible solutions given an initial starting point obtained by a constructive method. The first uses Variable Neighbourhood Descent, with an efficient method to generate new solutions which allows the algorithm to run relatively fast. Due to this property, given a time frame we can run it multiple times, similarly to the GRASP method. The second approach is based on Simulated Annealing. We propose a simple method to select initial parameters, based on the expected value of the objective function. The VND approach was superior both in terms of average deviation from best known solution, with 0.23%, and number of best solutions found, with 26, for a time frame of 40 minutes. We argue that this difference in performance is mainly related with how time-efficient the first solution is compared to the second. The two approaches could be further improved by trying to understand in which instances their solutions are not optimal. Additionally, a method to avoid evaluating the same initial solution twice could reduce the time to reach an optimal solution and increase diversification. Future work should aim in this direction.

References

Aickelin, Uwe. 2002. "An indirect genetic algorithm for Set Covering Problems". *Journal of the Operational Research Society* 53, no. 10 (october): 1118-26. https://doi.org/10.1057/palgrave.jors.2601317.

Baker, Edward K., Lawrence D. Bodin, William F. Finnegan, and Ronny J. Ponder. 1979. "Efficient heuristic solutions to an airline crew scheduling problem". *A I I E Transactions* 11, no. 2 (june): 79-85. https://doi.org/10.1080/05695557908974446.

Balas, Egon, and Maria C. Carrera. 1996. "A dynamic subgradient-based branch-and-bound procedure for Set Covering". *Operations Research* 44, no. 6: 875-90. https://doi.org/10.1287/opre.44.6.875.

Beasley, J. E. 1987. "An algorithm for Set Covering Problem". *European Journal of Operational Research* 31, no. 1 (july): 85-93. https://doi.org/10.1016/0377-2217(87)90141-X.

———. n.d. "OR-Library". Accessed December 03, 2018. http://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html.

Beasley, J. E., and P. C. Chu. 1996. "A genetic algorithm for the set covering problem". European Journal of Operational Research 94, no. 2 (october): 392-404. https://doi.org/10.1016/0377-2217(95)00159-X.

Crawford, Broderick, Ricardo Soto, Gino Astorga, and José García. 2018. "Constructive metaheuristics for the Set Covering Problem". In *Bioinspired Optimization Methods and Their Applications. BIOMA 2018*, edited by Peter Korošec, Nouredine Melab and El-Ghazali Talbi, 88-99. Lecture Notes in Computer Science. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-91641-5_8.

Dumais, Susan T. 2004. "Latent semantic analysis". *Annual Review of Information Science and Technology* 38, no. 1: 188-230. https://doi.org/10.1002/aris.1440380105.

Feo, Thomas A., and Mauricio G. C. Resende. 1989. "A probabilistic heuristic for a computationally difficult Set Covering Problem". *Operations Research Letters* 8, no. 2 (april): 67-71. https://doi.org/10.1016/0167-6377(89)90002-3.

Fortnow, Lance. 2009. "The status of the P versus NP problem". *Communications of the ACM* 52, no. 9 (september): 78-86. https://doi.org/10.1145/1562164.1562186.

Foster, B. A., and D. M. Ryan. 1976. "An integer programming approach to the vehicle scheduling problem". *Journal of the Operational Research Society* 27, no. 2: 367-84. https://doi.org/10.1057/jors.1976.63.

Gomes, Fernando C., Cláudio N. Meneses, Panos M. Pardalos, and Gerardo Valdisio R. Viana. 2006. "Experimental analysis of approximation algorithms for the vertex cover and Set

Covering Problems". *Computers & Operations Research* 33, no. 12 (december): 3520-34. https://doi.org/10.1016/j.cor.2005.03.030.

Hansen, Pierre, Nenad Mladenović, and José A. Moreno Pérez. 2010. "Variable neighbourhood search: methods and applications". *Annals of Operations Research* 175, no. 1 (february): 367-407. https://doi.org/10.1007/s10479-009-0657-6.

Karp, Richard M. 1972. "Reducibility among combinatorial problems". In *Complexity of Computer Computations*, edited by R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, 85-103. Boston, MA: Springer. https://doi.org/10.1007/978-1-4684-2001-2_9.

Kirkpatrick, Scott, C. D. Gelatt, and Mario P. Vecchi. 1983. "Optimization by Simulated Annealing". *Science* 220, no. 4598: 671-80. https://doi.org/10.1126/science.220.4598.671.

Ren, Zhi-Gang, Zu-Ren Feng, Liang-Jun Ke, and Zhao-Jun Zhang. 2010. "New ideas for applying ant colony optimization to the set covering problem". *Computers & Industrial Engineering* 58, no. 4 (may): 774-84. https://doi.org/10.1016/j.cie.2010.02.011.

Revelle, Charles, David Marks, and Jon C. Liebman. 1970. "An analysis of private and public sector location models". *Management Science* 16, no. 11: 692-707. https://doi.org/10.1287/mnsc.16.11.692.

Talbi, El-Ghazali. 2009. *Metaheuristics: From design to implementation*. Oxford: Wiley.

Yelbay, Belma, Ş. İlker Birbil, and Kerem Bülbül. 2014. "The set covering problem revisited: An empirical study of the value of dual information". *Journal of Industrial & Management Optimization* 11, no. 2: 575-94. https://doi.org/10.3934/jimo.2015.11.575.