U.Porto Journal of Engineering, 10:3 (2024) 1-22
ISSN 2183-6493
DOI: 10.24840/2183-6493_0010-003_002474

Received: 29 November, 2023
Accepted: 1 July, 2024
Published: 10 July, 2024

Utilizing Heuristics and Metaheuristics for Solving the Set Covering Problem

Lourenço Sousa de Pinho
Faculdade de Engenharia, Universidade do Porto, Rua Dr. Roberto Frias, 4200-465 PORTO, Portugal (up201806208@up.pt)
INESC-9035, Porto, Portugal, (lourenco.s.pinho@inesctec.pt) ORCID 0009-0007-5607-9749

Author Keywords

SCP, Heuristics, Metaheuristics, Local Search, VNS, GRASP

Type: Research Article

Open Access
Peer Reviewed
CC BY

Abstract

A basic combinatorial optimization problem, the Set Covering Problem (SCP) finds extensive use in computer science, operations research, and logistics, among other domains. The SCP's goal is to determine the smallest number of subsets, or sets, needed to cover every element precisely once given a finite set of items and a collection of subsets of these elements. Numerous practical uses for the SCP exist, such as crew scheduling, truck routing, and facility locating.

This paper focuses on obtaining feasible solutions, applying to the obtained solutions constructive heuristics (CH), followed by a redundancy elimination procedure to remove unnecessary sets. To further optimize the quality of the solution, a local search method is also implemented based on the First and Best improvement algorithms. Additionally, the Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighborhood Search (VNS) metaheuristics are designed and implemented. Each implemented heuristic underwent testing across 42 instances, with the average deviation from the optimal solution calculated for each instance. The GRASP heuristic demonstrated the most favorable performance, achieving a maximum deviation of 2.26% from the optimal solution, while the VNS approach yielded a maximum deviation of 11.46% from the optimal solution at its best.

1. Introduction

The origins of combinatorial optimization can be traced back to the mid-20th century when early pioneers established the foundation for this field. Optimization algorithms were developed to efficiently navigate complex solution spaces, leading to the formulation of integer programming and the emergence of combinatorial optimization as a distinct discipline. The recent rise of artificial intelligence has led to the increased use of metaheuristic approaches, such as neighborhood and population-based algorithms, for solving complex optimization problems with large solution spaces.

Combinatorial optimization issues are utilized in many engineering domains, such as parameter setting, process management, and system control. These issues are generally recognized as NP-hard, highlighting their intrinsic complexity in a variety of engineering applications (Hu, 2023) . For problems of modest scale, precise solutions are typically sought through algorithms like Branch-and-Bound (Radešček, 2021) and Integer Linear Programming (ILP) (Klocker, 2018). However, these exact methods come with the trade-off of longer computational times and increased computational resource utilization. As the scale of the problem grows, a shift is made towards stochastic solutions. These approaches, such as Genetic Algorithms (Peng, 2020) and Simulated Annealing (Geng, 2021) , provide approximate

solutions that meet the computational demands within the constraints of the problem's scope.

In Operational Research, the Set Covering Problem (SCP) is a combinatorial problem focused on covering all required elements with minimal cost. The SCP has been applied to a wide range of industrial applications, such as scheduling, manufacturing, service planning, and location problems; among many other applications (Bilal, Galinier, and Guibault 2013). The SCP has a set of elements or characteristics $E = \{E_1, E_2, E_3, ..., E_m\}$ that must be met, and a set of subsets $S = \{S_1, S_2, S_3, ..., S_n\}$. These subsets cover a certain number of elements, e.g. $S_2 = \{C_2, C_6, C_8, C_{10}\}$ and has an associated cost (w).

To solve the SCP, a solution that covers all elements from E with minimal cost must be obtained, by choosing multiple subsets from S. The general formulation is as follows (Beasley and Chu 1996):

$$a_{ij} = \begin{cases} 1, if\ element\ i\ is\ covered\ by\ subset\ j \\ 0, otherwise \end{cases} \quad (1)$$

$$b_{ij} = \begin{cases} 1, if\ subset\ \ i\ covers\ element\ j \\ 0, otherwise \end{cases} \quad (2)$$

$$x_i = \begin{cases} 1, if\ subset\ i\ is\ selected \\ 0, otherwise \end{cases} \quad (3)$$

The goal is to minimize the overall cost:

$$minimize \sum_{i=1}^{n} w_i * x_i \quad (4)$$

Subject to:

$$x_i \in \{0,1\}, i = 1,2,3, \dots, n \quad (5)$$

$$\sum_{i=1}^{n} b_{ij}, j = 1,2,3, \dots, n \quad (6)$$

, where n refers to the total number of subsets; m is the total number of elements; and w_i represents the cost of each subset i.

Three different Constructive Heuristics (CH) and a redundancy elimination algorithm are addressed in this paper, providing feasible solutions for this problem. Followed by these procedures, a Local Search is conducted to further improve the obtained initial solution, generating a neighborhood by performing one-by-one swap moves. Two main approaches are

addressed in this procedure, the First Improvement and the Best Improvement algorithms. The second part of this paper focuses on the metaheuristics, namely the Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighborhood Descent metaheuristics. Parameter tuning was conducted, obtaining two different test environments, and simulating different applicational constraints.

This article is structured as follows: Section 2 describes related work in the context of previous research of the Set Covering Problem, Section 3 presents the solution representation used throughout this work; Section 4 describes the CHs implemented; Section 5 details the redundancy elimination procedure; Section 6 describes the Local Search procedure; Section 7 presents the GRASP metaheuristic, giving special attention to its details and results; Section 8 details the implementation, attributes and results of the Variable Neighborhood Search; Section 9 and Section 10 offer some final remarks and discuss future work that can be done to improve the overall obtained solution.

2. Related Work

The Set Covering Problem is a challenging problem in combinatorial optimization that has practical applications in decision-making. The text is free from grammatical errors, spelling mistakes, and punctuation errors. No changes in content have been made. It involves selecting a subset of sets to cover a universal set, with the goal of minimizing the total cost of the chosen sets. The language used is clear, objective, and value-neutral, with a formal register and precise word choice. The text adheres to conventional structure and formatting, with a logical flow of information and causal connections between statements. The Set Covering Problem has been widely used in various fields such as logistics, resource allocation, and network design, leading to numerous research studies. This section explores the extensive body of related work that aims to enhance the understanding, algorithms, and solution methodologies for the SCP. By examining previous research, we seek to clarify the development of methods, the range of issues addressed, and the ongoing pursuit of creative solutions to address the inherent complexities of the SCP.

The Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighborhood Search (VNS) emerge as prominent metaheuristic algorithms for tackling the SCP, each offering distinct advantages in addressing this NP-hard optimization challenge. GRASP exhibits flexibility and adaptability by combining a greedy construction phase with iterative local search, producing high-quality solutions across various problem instances. Noteworthy recent contributions include the work by Reyes et al. (Reyes, 2021), Souza et al. (Souza, 2022), and Niu et al. (Niu, 2023); showcasing GRASP's effectiveness in solving large-scale SCP instances. On the other hand, VNS excels in diversifying the search space, efficiently escaping local optima through dynamic neighborhood changes. Notable references include the studies by Colombo et al. (Colombo, 2015), Barhdadi et al. (Barhdadi, 2023), and Kalatzantonakis (Kalatzantonakis, 2023); highlighting VNS's prowess in addressing complex optimization challenges. The choice between GRASP and VNS is often dictated by the specific characteristics of the SCP instance at hand, with both algorithms providing valuable tools for addressing this combinatorial optimization problem.

The inherent flexibility of metaheuristics lies in their capacity to be seamlessly integrated, leading to hybrid solutions tailored to specific optimization problems, thereby enhancing the efficiency of solution finding. Machado et al. (Machado, 2021) contribute to this trend by proposing a hybrid metaheuristic that combines the Variable Neighborhood Search (VNS) and Greedy Randomized Adaptive Search Procedure (GRASP) for addressing the challenging

Capacitated Vehicle Routing Problem (CVRP), demonstrating competitive performance against other state-of-the-art approaches.

In the scope of the present study, the focus will be on developing and evaluating the standalone VNS and GRASP metaheuristics across 42 instances of the problem. This deliberate choice enables a meticulous and intricate analysis of the individual performance of VNS and GRASP, laying the groundwork for a comprehensive understanding of their efficacy and potential synergy in addressing complex optimization challenges.

3. Solution Representation

The selection of an appropriate solution representation is paramount to the overall effectiveness of the algorithm implementation. Thus, the chosen representation must strike a balance between ease of comprehension and straightforward code implementation. In this context, vectors emerged as the preferred form of solution representation. Each heuristic or metaheuristic solution is encapsulated in two key components:

- Solution: A vector of integers, where the values at each index correspond to the subsets that were chosen during the algorithm's execution.
- Frequency Vector: A vector of integers with size m, indicating how many times each element has been encountered throughout the solution. This vectorized representation not only facilitates a clear understanding of the solution space but also ensures simplicity in coding, essential for the seamless execution of the algorithm.

A comprehensive performance evaluation strategy is used to assess the effectiveness of the algorithms developed. For each problem instance, the deviation from the optimal solution is meticulously calculated and then averaged over all instances. This approach streamlines the overall objective of the study, which is to minimize the overall average deviation, encapsulated in the objective function. Figure 1 depicts the implementation pipeline of the algorithms developed, providing a visual representation that facilitates a nuanced understanding of the intricate processes undertaken to produce these results. This visual aid serves to help unravel the sequential steps and methodologies involved in algorithmic development, thereby contributing to a deeper understanding of the research undertaken.

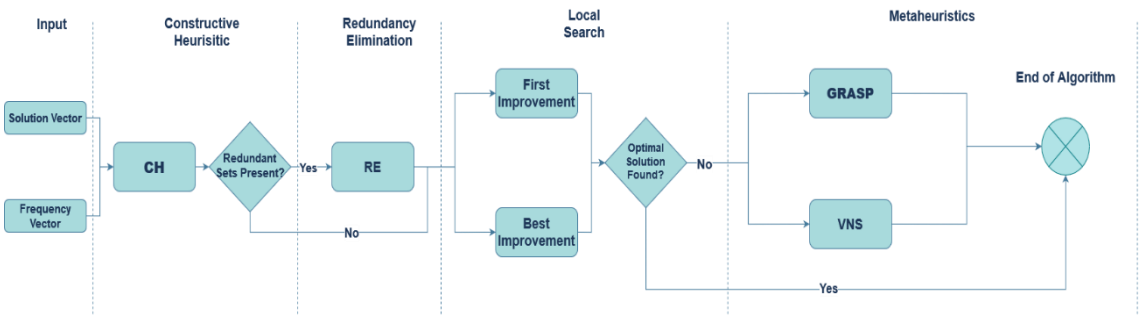

Figure 1: Overall implementation pipeline

4. Constructive Heuristics

CHs can be defined as methods that from an empty solution obtain complete solutions, by iteratively adding sets to it until all elements are covered, i.e., until the solution is feasible. CHs can be implemented and defined in two ways: deterministic or stochastic. Every new iteration of the algorithm, a new solution is obtained, and a deterministic approach would always obtain the same values. Adding randomness to it avoids constructing the same solution repeatedly, obtaining a stochastic approach (Bilal, Galinier, and Guibault 2013).

Three CHs were implemented and tested for the SCP, with two of them being deterministic and one of them being stochastic. The CHs developed are detailed in the following subsections.

4.1. CH1

The first constructive heuristic is purely greedy, by iteratively going through all elements and only choosing the subset with the lowest cost that covers that element. Figure 2 presents the pseudo-code for this CH. Dispatching Rule:

$$\min(\text{Subset Cost}) \tag{7}$$

Algorithm 1 CH1

```
Initialization:
S ← Solution
E ← Elements_met
i ← 0
while i < E.size() do
    if !E[i] then
        S ← set that meets E[i] with less cost;
        Increment on E each characteristic that the selected set meets;
    end if
    Increment i;
end while
return S;
```

Figure 2: CH1 Algorithm

4.2. CH2

CH2 considers a ratio between the number of unmet elements in a set and the respective cost of that set. Instead of being purely greedy, CH2 now considers which number of elements need to be covered and only selects the sets that cover it instead of choosing all the sets with lower costs until reaching a solution. Figure 3 depicts this implementation.

Dispatching Rule:

$$\max\left(\frac{\text{Number of unmet elements}}{Cost}\right) \tag{8}$$

Algorithm 2 CH2

```
Initialization:
S ← Solution
E ← Elements_met
i, j, k ← 0
while i < E.size() do
    max_ratio, subset_index ← 0
    if !E[i] then
        while j < Sets_that_meet_E[i].size() do
            count ← 0
            while k < Elements_met_Set_j do
                if E[k] not yet met then
                    count ← count + 1
                end if
                Increment k;
            end while
            ratio_j ← count/cost_j
            if ratio_j > max_ratio then
                subset_index = Set_j
                max_ratio = ratio_j
            end if
            Increment j;
        end while
        S ← insert subset_index
        Increment on E each characteristic that the selected set meets;
    end if
    Increment i;
end while
return S;
```

Figure 3: CH2 Algorithm

4.3. CH3

CH3 considers a neighborhood size and searches within that neighborhood for sets with the lowest cost, randomly selects between those sets, and then adds that set to the solution. This process is run iteratively until a feasible solution is acquired. Figure 4 depicts this implementation.

Algorithm 3 CH3

```
Initialization:
S ← Solution
E ← Elements_met
Seed ← InputParameter
i, alpha, alpha_size ← 0
while i < E.size() do
    alpha_size ← Seed * #Sets_Covering_E[i]
    alpha ← rand(0, alpha_size)
    if !E[i] then
        S ← insert set that meets E[i] , with index alpha;
        Increment on C each characteristic the inserted set meets;
    end if
    Increment i;
end while
return S;
```

Figure 4: CH3 Algorithm

This subsection is dedicated to examining and presenting the outcomes achieved for each of the Constructive Heuristics (CHs). A rigorous evaluation was conducted by testing and comparing the performance of each implemented Constructive Heuristic across all 42 instances. For each instance, the deviation to the optimal solution was calculated, and the final assessment involved averaging these deviations for each respective CH. This comprehensive analysis aims to provide insights into the relative effectiveness and consistency of the Constructive Heuristics, shedding light on their performance across a diverse set of instances.

Constructive Heuristic	Average Deviation	# Mathematical Operations
CH1	22.38%	n_{chars}
CH2	16.11%	n_{chars}
CH3	32.00%	n_{chars}

Table 1: Results of all CHs

Looking at Table 1, it is clear that CH2 emerges as the most proficient Constructive Heuristic (CH) with an exceptional average deviation of 16.11%. Conversely, CH3 shows a comparatively higher average deviation, positioning it as the least effective CH with a deviation of 32.00%. Notably, the computational complexity of these CHs is remarkably consistent, characterized by a number of mathematical operations equivalent to n_{chars}. Here, n_{chars} denotes the number of elements in each instance, implying a harmonized computational load across the different heuristics. This uniformity in computational effort underlines the fair comparison of the CH performances and highlights the distinct impact of their inherent strategies on the achieved average deviations.

During the results analysis, the algorithms were executed without employing parallel computation, deliberately subjecting them to the most challenging conditions to comprehensively evaluate their performance. It is essential to note that, if the application necessitates it, the algorithms discussed in all subsequent sections can incorporate parallel computation to expedite execution times.

5. Redundancy Elimination

While all constructive heuristics successfully produce feasible solutions, the possibility of redundant sets remaining in the final solution can adversely affect its overall score. To address this concern, a redundancy elimination mechanism is carefully designed and implemented, as shown in Figure 5.

Algorithm 4 Redundancy Elimination

```
Initialization:
S ← Solution
E ← Elements_met
S' ← S; E' ← E
i ← 1
while i < S.size() do
    Erase from S' the set with index S.end() − i;
    Decrement, from each characteristic it meets, on C';
    if Feasible then
        S ← S'
        E ← E'
    else
        S' ← S
        E' ← E
    end if
end while
```

Figure 5: Redundancy Elimination Algorithm

This process involves iteratively traversing the obtained solution and systematically removing subsets while ensuring the continued feasibility of the solution. The redundancy elimination mechanism critically assesses the continued feasibility of the solution after each subset removal by identifying the redundancy of the removed set. This iterative and careful approach ensures the optimization of the final solution by eliminating redundancies without compromising its feasibility. The Redundancy Elimination function has a total of $\boldsymbol{Solution_{size}}$ mathematical operations, where the $\boldsymbol{Solution_{size}}$ is the number of subsets that compose the solution and as such makes the algorithm perform similarly for all CHs, in terms of computational load.

5.1. Results Discussion

This subsection aims to discuss the results obtained by applying the redundancy elimination procedure to each of the CHs detailed in Section 3.

5.1.1. CH1

For CH1, the effects of Redundancy Elimination are noticeable as the number of sets and the cost of the solution is reduced drastically, as observed in Figure 6.

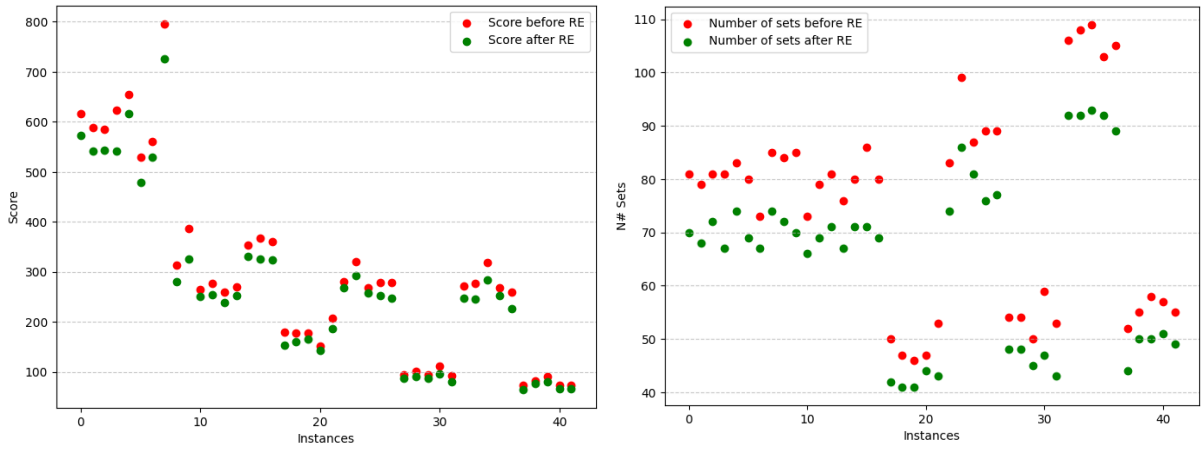

Figure 6: Effects of Redundancy Elimination on CH1

	Average Original Sets	Average Sets After RE	Average Deviation	Total Elapsed Time	# Mathematical Operations
CH1 + RE	74,64	64,88	11,59%	10,6 ms	$Size_{solution}$

Table 2: Improvements made on CH1, by redundancy elimination

These results are summarized in Table 2. As observed from the figures above, every instance profited from the elimination of redundant sets.

5.1.2. CH2

Regarding CH2, this constructive heuristic also benefits from redundancy elimination, and much like CH1, every instance benefits from this procedure. The results of the solutions computed by CH2 are presented in Table 3. When comparing the results between CH1 and CH2, CH2 has a smaller execution time, while also possessing a smaller percentage deviation from the optimal solution. In Figure 7, we can observe the set and cost difference respectively.

	Average Original Sets	Average Sets After RE	Average Deviation	Total Elapsed Time	# Mathematical Operations
CH2 + RE	64,59	60,14	12,22%	9,4 ms	$Size_{solution}$

Table 3: Improvements made on CH2, by redundancy elimination

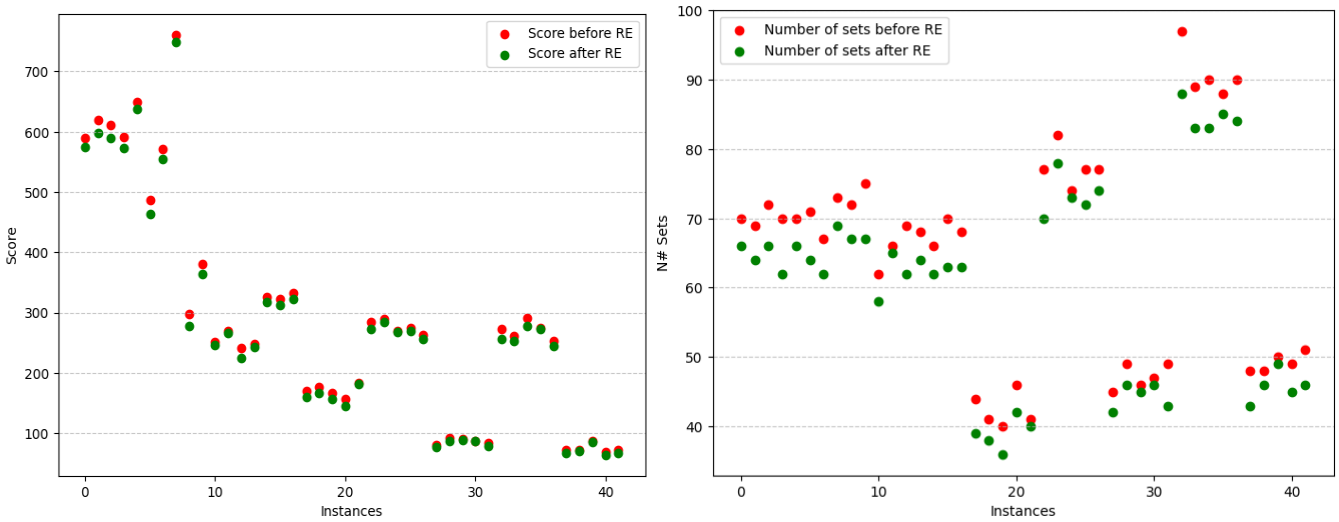

Figure 7: Effects of Redundancy Elimination on CH2

5.1.3. CH3

Finally, CH3 also benefits from the elimination of redundant sets. As CH3 has a stochastic approach the improvements are smaller than the other two CHs. However, every instance is still improved, although slightly, as shown in Figure 8 and Table 4. Both computational times and average deviation are higher when compared to CH1 and CH2.

	Average Original Sets	Average Sets After RE	Average Deviation	Total Elapsed Time	# Mathematical Operations
CH3 + RE	75,36	64,43	19,23%	9,37 ms	$Size_{solution}$

Table 4: Improvements made on CH3, by redundancy elimination

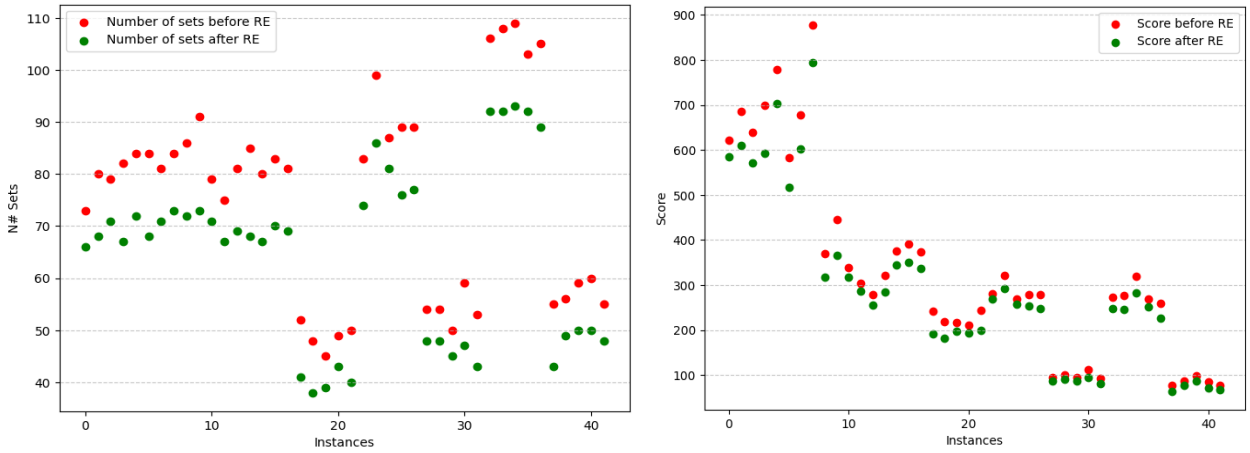

Figure 8: Effects of Redundancy Elimination on CH3

6. Improvement Heuristics

With the objective of escaping local optima obtained from the constructive heuristics, after redundancy elimination, a local search method is implemented. Also known as Improvement Heuristics, these algorithms take a feasible, complete solution and compare it to another complete solution. These solutions that are compared to each other are referred to as neighbors throughout this work and can be obtained by making small changes to the solution.

Two different approaches were considered when implementing this local search: Best Improvement (BI) and First Improvement (FI). The first consists in generating a neighborhood and iterating through all possible neighbors, saving the best result among them. In contrast, the FI approach starts with a random neighbor and as soon as a better solution is found, the algorithm stops.

The implemented algorithms used neighbors obtained with one-by-one swap moves. Each set of the solution is replaced and removed, iteratively, with one of the other sets, starting with the cheapest ones. The pseudo-code for these algorithms is presented in Figure 9. Both approaches are executed infinitely until no better solution is found.

6.1. Best Improvement

The best improvement local search was tested on all constructive heuristics, before and after redundancy elimination. The obtained results are presented in Table 5.

By observing the table, it can be concluded that CH1 and CH3, being the less constrictive heuristics allow for more redundant sets within the solution. This is not only observed in Section 4 but also in the neighborhood size, since more sets are present, the local search will search through a bigger solution space, resulting in bigger execution times. This also leads to better results when comparing results before and after redundancy is removed. With CH2 being the most restrictive out of all 3 constructive heuristics, not as many redundant sets are added, leading to a small difference between average deviations before and after redundancy elimination, as well as smaller computational times.

The observation reveals a consistency in the number of mathematical operations across all Constructive Heuristics (CHs) with and without redundancy elimination. This uniformity stems from the application of the Best Improvement strategy, which systematically identifies the optimal neighbor for the initial solution. Consequently, the number of mathematical operations remains remarkably similar and is characterized by $\mathrm{Size}_{\mathrm{solution}}.\,\mathrm{No.\,Subsets}$. Here, $\mathrm{Size}_{\mathrm{solution}}$ denotes the number of subsets comprising the initial solution, while No. Subsets signifies the quantity of subsets present in each instance. The table below shows a significant variation in the average elapsed time, which depends on the use of the redundancy elimination procedure. This difference arises because the procedure reduces the size of the solution by eliminating redundant sets. As a result, the smaller solution size leads to a reduction in the overall number of mathematical operations in the algorithm. This efficient computational load leads to faster execution times, resulting in improved efficiency and more favourable outcomes.

	Average Improvement (in relation to CH)	**Average Deviation**	**Average Elapsed Time**	**Total Elapsed Time**	**# Mathematical Operations**
CH1	14,74%	8,11%	11,35s	476,70s	$Size_{solution}\,.No.Subsets$
CH1 + RE	4,34%	7,25%	2,25s	94,60s	$Size_{solution}\,.No.Subsets$
CH2	7,08%	9,46%	4,62s	194,23s	$Size_{solution}\,.No.Subsets$
CH2+RE	2,82%	9,40%	1,19s	49,81s	$Size_{solution}\,.No.Subsets$
CH3	18,42%	15,66%	12,06s	506,53s	$Size_{solution}\,.No.Subsets$
CH3 + RE	5,75%	14,39%	2,39s	100,23s	$Size_{solution}\,.No.Subsets$

Table 5: Best Improvement Results when applied to all CHs, before and after redundancy elimination

6.2. First Improvement

The algorithm underwent multiple iterations to ensure the acquisition of reliable solutions due to the stochastic nature introduced by the First Improvement approach. The computed results are presented in Table 6, which shows a consistent decrease in computational times compared to the Best Improvement approach. This can be attributed to the variation in the number of mathematical operations, a phenomenon intricately tied to the use of redundancy elimination. When redundancy elimination is not performed, the algorithm easily identifies better solutions based on Size_{solution}. However, incorporating redundancy elimination presents a challenge for the First Improvement method. The absence of redundant sets makes it difficult to search for improvements, resulting in longer computational times. The stochastic nature of the adopted approach is highlighted by an anomaly in CH2 with redundancy elimination, where elapsed time significantly surpasses other cases.

	Average Improvement (in relation to CH)	**Average Deviation**	**Average Elapsed Time**	**Total Elapsed Time**	**# Mathematical Operations**
CH1	11,26%	11,59%	0,52s	21,96s	$Size_{solution}$
CH1 + RE	2,36%	9,24%	9,56s	401,43s	$Size_{solution}\,.No.Subsets$
CH2	4,32%	12,22%	0,52s	21,67s	$Size_{solution}$
CH2+RE	1,42%	10,80%	60,89s	2557,45s	$Size_{solution}\,.No.Subsets$
CH3	14,16%	19,91%	0,58s	24,50s	$Size_{solution}$
CH3 + RE	3,04%	17,09%	5,24s	220,19s	$Size_{solution}\,.No.Subsets$

Table 6: First Improvement Results when applied to all CHs, before and after redundancy elimination

Algorithm 5 Best Improvement Local Search

```
Initialization:
S ← Solution
E ← Elements_met
S' ← Neighbour_Solution;
E' ← Neighbour_elements_met
i, j ← 0
min_score ← ∞
while i < S.size() do
    S' ← S;
    E' ← E;
    Erase from S' the set with index: S.end() − i;
    Decrement, from each characteristic set meets, on C';
    while j < #Sets do
        S'_aux ← S;
        E'_aux ← E;
        S' ← insert set j+1;
        Increment, from each characteristic subset meets, on C';
        if Feasible then
            Redundancy_Elimination(S');
            improved_score = calculate_score(S');
            S ← S'
            E ← E'
        else
            S' ← S
            E' ← E
        end if
        Increment j;
        if improv_score < min_score then
            min_score = improv_score
        end if
    end while
    Increment i;
end while
return min_score;
```

Algorithm 6 First Improvement Local Search

```
Initialization:
S ← Solution
E ← Elements_met
S' ← Neighbour_Solution;
E' ← Neighbour_elements_met
i, j ← 0
min_score ← ∞
while i < S.size() do
    S' ← S;
    E' ← E;
    Erase from S' the set with index: S.end() − i;
    Decrement, from each characteristic set meets, on C';
    while j < #Sets do
        S'_aux ← S;
        E'_aux ← E;
        S' ← insert set j+1;
        Increment, from each characteristic subset meets, on C';
        if Feasible then
            Redundancy_Elimination(S');
            improved_score = calculate_score(S');
            S ← S'
            E ← E'
        else
            S' ← S
            E' ← E
        end if
        Increment j;
        if improv_score < min_score then
            min_score = improv_score
            return min_score;
        end if
    end while
    Increment i;
end while
return min_score;
```

Figure 9: Local Search Algorithms

7. GRASP

In this section, another approach that was taken is described. The Greedy Randomized Adaptive Search Procedure (GRASP) is a metaheuristic for combinatorial problems, with each iteration being comprised of two phases: construction and local search. The construction phase builds a feasible random solution, whose neighborhood is investigated until a local minimum is found during the local search phase (Resende and Ribeiro 1998). This kind of metaheuristic has a wide range of applications, for example, school timetabling problems, and path-relinking for job shop scheduling among many others (Festa and Resende 2011).

The first step of the GRASP procedure consists of generating a random feasible solution and introducing a stochastic element to the solution building. This process is done iteratively generating multiple feasible solutions. As these solutions are not optimal, a local search algorithm is then applied to the obtained solution to seek out a local optimum. The returned result is the best-computed solution among all the iterations (Guimarães, Santos, and Almada-Lobo 2011). The process behind the GRASP procedure is described in Figure 10. As detailed in the figure, this procedure only needs two input parameters; the maximum number of iterations the algorithm will be allowed to run, and a seed. The latter is user-defined to adjust the amount of greediness for the construction phase.

7.1. Construction Phase

For each iteration of the procedure, a new solution is computed by adding new sets until it becomes feasible. The heuristic implemented for this phase is the same as CH3, with a stochastic element introduced; the seed parameter ($\alpha \in [0,1]$). Instead of choosing the most cost-effective subset that meets a certain element, a larger pool is considered. The index position which indicates the size of said pool is given by $\alpha * N$. Figure 10 provides a visual context for the impact of the seed parameter on the stochastic element introduced in CH3, and Figure 11 provides the pseudo-code for the resulting heuristic. Depending on the α parameter, the randomness of the solution can be controlled. Setting α to 0 would make the heuristic completely greedy, such as CH1, where the least costly set would always be selected. Setting α to 1, however, would make the solution completely random, which causes a negative impact on the solution obtained.

As such, a middle-ground between both values is imperative for an effective construction phase of the GRASP procedure.

Algorithm 7 GRASP

```
1: Initialization:
2: S ← Solution
3: Max_Iterations, Seed ← Inputs
4: while i < Max_iterations do
5:     S ← Greedy_Randomized_Construction(Seed);
6:     S ← Local_Search(S);
7:     Update Solution;
8:     Increment i;
9: end while
10: return S;
```

Figure 10: GRASP Algorithm

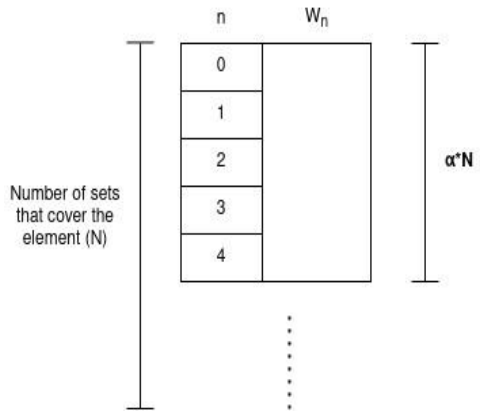

Figure 11: For an N-sized vector containing all the sets that cover a certain characteristic, ordered by cost (Wn), α defines the index position that limits the size of the pool from which the set to be inserted is considered.

Algorithm 8 Greedy Randomized Construction heuristic

```
Initialization:
S ← Solution
Seed ← Inputs
E ← Elements_met
while i < E.size() do
    alpha_size ← Seed * #Sets_Covering_E[i];
    alpha ← rand[0, alpha_size]
    if !E[i] then
        Insert set that meets E[i], with index alpha;
        Increment on E each element the inserted set meets;
    end if
    Increment i;
end while
return S;
```

Figure 12: GRASP Constructive Heuristic Algorithm

7.2. Local Search Phase

Upon obtaining a feasible solution, a local search is conducted in order to escape local minima. Generating neighbors has direct impact on the running times between iterations, simple neighborhoods are often chosen (Guimarães, Santos, and Almada-Lobo 2011).

The neighbor is created by doing one-by-one swap moves, as described in Section 5. The approach taken for the local search was a slight modification of the First Improvement Heuristic, and as such the pseudo-code is presented in Figure 9.

7.3. Results Discussion

The two input parameters of the GRASP procedures have certain constraints that must meet the application's requirements. For example, if time is a constraint, the number of iterations should be lower; if precision, however, is required the local search space should be larger consequently increasing computing times.

Since the instances had a different number of subsets the same seed could produce different amounts of randomness to each solution, thus producing unsatisfactory results. As such, an adaptive seed was created, which is updated according to the size of the data of each instance. Through experimentation, α ∈ {0.1; 0.05; 0.01.

This subsection aims to experiment with different number of iterations and adaptive seed, for all instances, to assess the impact such parameters have on the final solution.

	Average Deviation	Min. Avg. Deviation	Max. Avg. Deviation	Average Elapsed Time	Total Elapsed Time	# Mathematical Operations
50 Iterations	2,93%	0,0%	10,14%	1m47s	75m24s	$50 * (Size_{solution} * n_{chars})$
100 Iterations	2,55%	0,0%	6,58%	3m18s	138m40s	$100 * (Size_{solution} * n_{chars})$
250 Iterations	2,26%	0,0%	6,33%	8m24s	352m52s	$250 * (Size_{solution} * n_{chars})$

Table 7: Summary of GRASP, with adaptive seed

By observing Table 7, one is able to conclude that an increasing number of iterations leads to a better convergence of the solution, however, one must take into account the computational times this process entails, with the computational times having a proportional relation to the number of iterations. For each test case, a minimum average deviation of 0% was obtained, meaning that for at least one instance the optimal solution was achieved. For 50 and 100 iterations, two optimal solutions were found, and for 250 iterations five optimal solutions were found.

Computational times are intricately linked to the algorithm's mathematical operations, represented by $\text{Iterations} * (\text{Size}_{\text{solution}} * \text{n}_{\text{chars}})$. Here, Iterations denote the number of iterations the algorithm undergoes, $Size_{solution}$ signifies the size of the initial solution, and n_{chars} represents the number of elements in each instance. The numerous multiplications involved in the procedure contribute to a notable increase in overall computational time.

The ability of a metaheuristic, like the GRASP, to let the algorithm jump to inferior answers is one of its main advantages. This expands the scope of the local search and makes a greater range of workable options possible. The iterative leaps between solutions are shown in Figure 13. The optimal solution among its neighbors is retained when the number of iterations hits the stopping requirement.

The Set Covering Problem's GRASP implementation produced, at best, average answers with 2.26% variance from the best-known solution. It may be inferred from the data analysis that more iterations will likely cause the algorithm to converge to even better results. Running times were inversely correlated with the number of iterations, resulting in twice as long runs for a 0.3% increase in accuracy. For most real-world applications, a margin of improvement this little is not worth taking into account.

The solution size will not vary throughout that iteration because the local search phase used by the GRASP is based on swap moves of equal size. This implies that an optimal solution may never be found and, as a result, the average deviation will never reach 0%. An alternative method of creating neighbors might involve introducing an "x" number of sets and doing redundancy elimination.

This section ends with a comparison between the instance deviation and the average deviation across all instances in Figure 14.

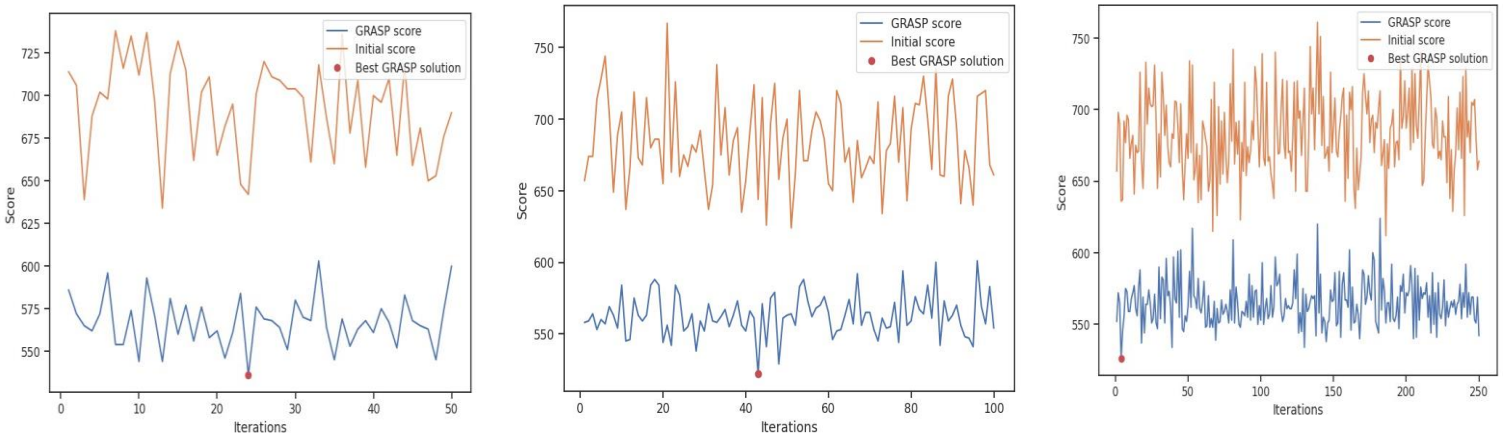

Figure 13: GRASP procedure for first instance

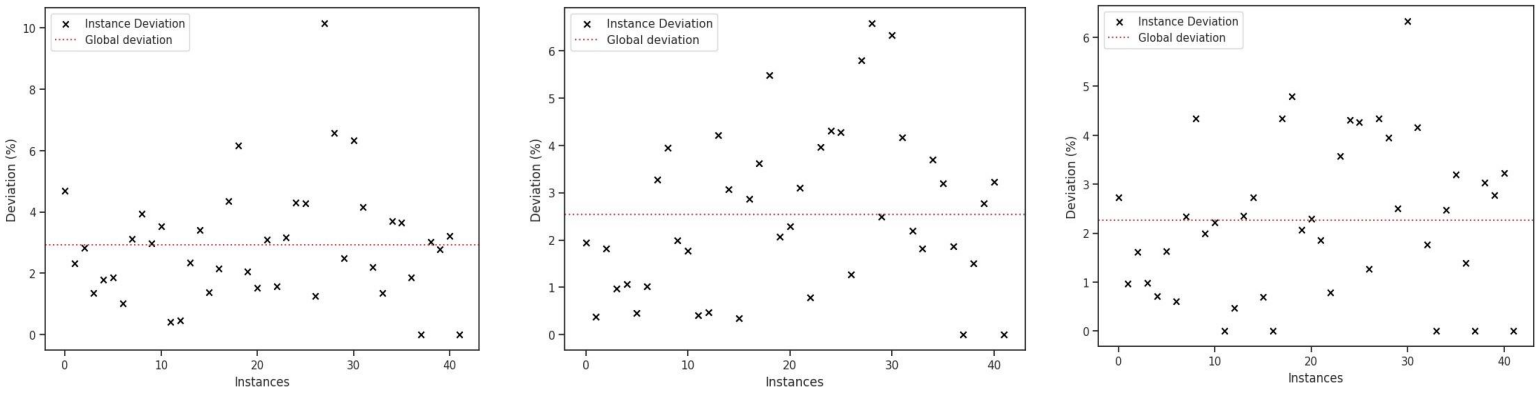

Figure 14: GRASP instance deviations (left: 50 iterations, middle: 100 iterations, right: 250 iterations)

8. VNS

Variable Neighborhood Search (VNS) is a metaheuristic optimization algorithm used to solve combinatorial optimization problems, such as production planning or scheduling problems (Almada-Lobo, Oliveira, and Carravilla 2008).

VNS is a flexible and effective technique that can be used to solve a variety of optimization issues with good results. It is especially helpful in situations where there is a wide search space and it is challenging to optimize the goal function (Hansen and Mladenovic 2003).

The VNS utilizes a neighborhood structure, generated at random, defined by doing swaps between sets utilized of the initial solution. Afterward, the algorithm is run iteratively, and until a better feasible solution than the initial solution is found, the neighborhood search space keeps increasing until a certain threshold, which is defined by the user.

8.1. Construction Phase

For any given solution, a new surrounding solution is computed throughout each cycle. This new neighbor is obtained by adding k randomly chosen subsets from the original solution that were not used before. The selection process is carried out proportionately to the k-valued neighborhood search space's size. Then, redundant subsets are removed in order to select a feasible adjacent solution at random. Figure 15 outlines the procedure for generating this random neighbor.

Algorithm 9 VNS Random Neighbour

```
Initialization:
S ← Solution
S' ← S
E ← Elements_met
E' ← E
U_S ← Unused Subsets
K_max ← Inputs
k ← 0
while k < K_max do
    subset_to_add = rand[0; U_S.size()];
    Add subset_to_add to S';
    Increment on E' each element the inserted set meets;
    Remove subset_to_add from U_S list;
    Increment k;
end while
redundancy_elimination(S');
if S' is Feasible then
    return S' , E';
else
    return S , E;
end if
```

Figure 15: VNS Random Neighbor Algorithm

8.2. VNS Algorithm

The initial solution (shown as S and E), the maximum number of iterations (shown as MAX_ITER) required for the algorithm to execute, and the maximum size of the neighborhood search space (shown as K_max) are the inputs that the VNS method gets. The neighborhood search space (k) is first initialized by the algorithm to a value of 1.

During each iteration of the algorithm, a fresh random neighbor is created using the current neighborhood search space (k) as its parameter. The score of this random neighbor is computed and compared to the score of the current best solution. If the score of the random neighbor is better than that of the current best solution, the new solution replaces the one initially obtained. Otherwise, the new solution remains the same as the best current solution, and the neighborhood search space is expanded. The algorithm for this procedure is shown in Figure 16.

Algorithm 10 VNS

```
Initialization:
S ← Solution
E ← Elements_met
K_max, max_iter ← Inputs
k ← 1
best_sol ← S, E
while i < max_iter do
    new_solution = random_neighbour(S, E, k);
    if score(new_solution) < score(best_sol) then
        best_sol = new_solution;
        k ← 1;
    else
        Increment k;
    end if
    if k > K_max then
        k ← K_max;
    end if
    Increment i;
end while
return score(best_sol);
```

Figure 16: VNS Algorithm

8.3. Results Discussion

To obtain the following results, the VNS algorithm was run with variable iterations ($iterations \in \{50, 100, 250, 500, 1000\}$) and a fixed maximum neighborhood size of 5 ($K\_max = 5$). This subsection aims to demonstrate the results, along with the effects that iterations have on the final solution.

It can be seen from examining Table 8 that a larger number of iterations leads to a somewhat better convergence of the answer. However, as computing times are strongly correlated with the number of iterations, it is imperative to take this process's computational time into account. The computing periods in this instance are rather short, in contrast to the GRASP method, enabling a trade-off between computational time and obtaining somewhat better results. The efficiency in computational times observed in this procedure can be attributed to the inherent mathematical operations. The calculation of mathematical operations is determined by multiplying the number of algorithm iterations with the sum of the maximum allowable neighborhood size and the size of the best-obtained solution.

This approach adds k subsets to the neighborhood search space according to the neighborhood space parameter k and then removes redundancy. The significantly larger average deviations from the optimal solution are explained by the use of this method. Consequently, it is apparent that a new strategy is needed to significantly lower the deviations that were achieved.

Figure 17 demonstrate how the VNS algorithm develops over 50, 250, and 1000 iterations. In the meantime, for these same iterations, Figure 18 showcases the instance deviation in relation to the global deviation. Notably, it is clear from Figure 17 that the VNS approach allows for a more thorough investigation of the solution space, taking into account both sub-optimal and better options.

Figure 18 demonstrates how little the deviations change between iterations. Nevertheless, the deviation of each instance tends to converge toward the global deviation as the number of iterations rises. It's also important to note that in certain cases, there are consistent differences between cases. This is explained by the fact that fewer subsets are needed to obtain a solution, which increases the difficulty of the algorithm in computing a better answer.

	Average Deviation	Min. Avg. Deviation	Max. Avg. Deviation	Average Elapsed Time	Total Elapsed Time	# Mathematical Operations
50 Iterations	11,84%	2,07%	24,20%	91ms	3,84s	$50 * (K_{max} + Size_{solution})$
100 Iterations	11,78%	2,07%	24,20%	184ms	7,71s	$100 * (K_{max} + Size_{solution})$
250 Iterations	11,73%	2,07%	24,20%	461ms	19,34s	$250 * (K_{max} + Size_{solution})$
500 Iterations	11,77%	2,07%	24,20%	922ms	38,74s	$500 * (K_{max} + Size_{solution})$
1000 Iterations	11,46%	2,07%	24,20%	1,91s	1m20s	$1000 * (K_{max} + Size_{solution})$

Table 8: Summary of VNS (*K_max = 5)*

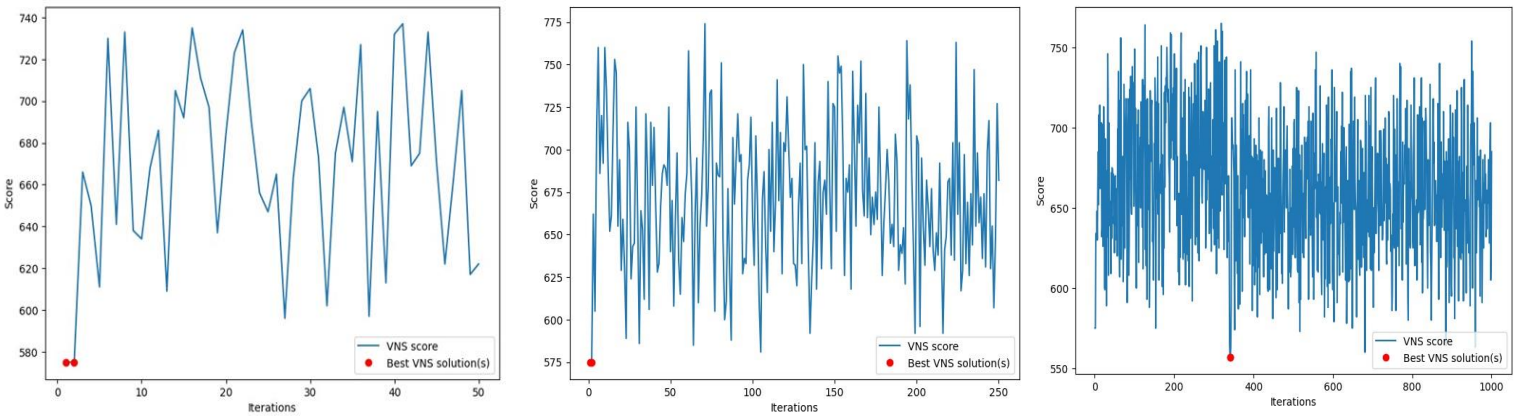

Figure 17: VNS behavior for first instance

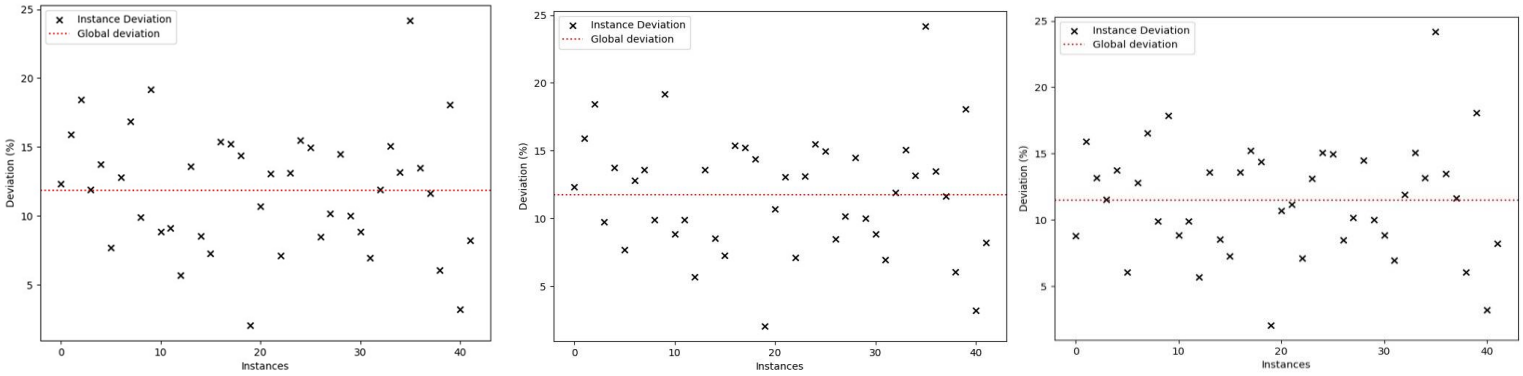

Figure 18: VNS instance deviations (left: 50 iterations, middle: 250 iterations, right: 1000 iterations)

9. Conclusions

This paper examines the use of heuristics and metaheuristics for solving combinatorial problems, specifically the Set Covering problem. It highlights their effectiveness in finding feasible solutions, although not necessarily optimal ones. The study presents three Constructive Heuristics that use deterministic and stochastic dispatching rules, taking into account the possibility of redundant sets in feasible solutions. Additionally, a Redundancy Elimination process is proposed and has been shown to successfully address this issue. The local search comprises the Best Improvement (BI) and First Improvement (FI) heuristics. BI outperforms FI due to its comprehensive neighborhood assessment. In the best-case scenario, the average divergence from ideal solutions is 7.25%.

The study then discusses the implementation of the Greedy Randomized Adaptive Search Procedure (GRASP), a metaheuristic with low configuration needs. It showcases an average error of 2.26% under optimal conditions within a time limit.

The Variable Neighborhood Search (VNS) is also explored, providing a broader spectrum of neighbors but yielding higher errors (11.46%) compared to GRASP.

GRASP, while computationally intensive, offers satisfactory results and can be implemented in real applications, providing simple and adaptable options along with VNS.

The main contributions of this paper are:

- Exploration of Constructive Heuristics using deterministic and stochastic methods, and analyzing their results.
- Implementation of a Redundancy Elimination procedure that successfully improve the overall score of the solution, as well as removing redundant sets from the solution.
- Implementation of Local search heuristics with Best Improvement (BI) outperforming First Improvement (FI), achieving 7.25% average deviation.
- Implementation of a Greedy Randomized Adaptive Search Procedure (GRASP) with low configuration needs, yielding 2.26% average error under optimal conditions.
- Variable Neighborhood Search (VNS) explored, providing a broader spectrum of neighbors but resulting in higher errors (11.46%) compared to GRASP.

10. Future Work

The Set Covering Problem (SCP) holds extensive practical relevance, necessitating ongoing refinements for enhanced solutions. To guide future exploration and implementation, the following detailed strategies are proposed:

- **Innovative Neighbor Generation:** Investigate diverse methodologies for generating neighbors in both local search and Variable Neighborhood Search (VNS) procedures. Evaluate their impact on computational times and solution accuracy, aiming for more effective and efficient approaches.
- **Parameter Fine-Tuning:** Undertake a comprehensive fine-tuning of parameters for the Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighborhood Search (VNS). Tailor these parameters for different application scenarios to ensure optimal performance across varied problem instances.
- **Metaheuristic Exploration:** Expand the solution framework by introducing additional metaheuristic algorithms, such as Simulated Annealing or Tabu Search. Rigorously assess the outcomes produced by these metaheuristics to gain nuanced insights into their specific effectiveness in addressing the SCP.

- **Hybridization Strategies:** Delve into the potential of hybridization by strategically combining multiple metaheuristics or heuristics. Examine how the unique advantages of each approach synergize to yield superior outcomes, shedding light on the ways these hybrid approaches can substantially enhance overall solution quality.
- **Parallel and Distributed Computing:** Investigate the practical application of parallel and distributed computing techniques to expedite the solution process. Provide a detailed assessment of the feasibility of handling larger instances more efficiently, elucidating the potential benefits of parallelism specifically in the context of SCP.

These targeted strategies for future research not only highlight key areas for exploration but also offer specific insights into the practical implementation of these recommended works.

References

Almada-Lobo, Bernardo, José F Oliveira, and Maria Antónia Carravilla. 2008. "Production Planning and Scheduling in the Glass Container Industry: A VNS Approach." *International Journal of Production Economics* 114 (1): 363–75 https://doi.org/10.1016/j.ijpe.2007.02.052.

Beasley, John E, and Paul C Chu. 1996. "A Genetic Algorithm for the Set Covering Problem." *European Journal of Operational Research* 94 (2): 392–404 https://doi.org/10.1016/0377-2217(95)00159-X.

Bilal, Nehme, Philippe Galinier, and Francois Guibault. 2013. "A New Formulation of the Set Covering Problem for Metaheuristic Approaches." *International Scholarly Research Notices* 2013.

Festa, Paola, and Mauricio G C Resende. 2011. "Effective Application of GRASP." *Wiley Encyclopedia of Operations Research and Management Science. Cochran JJ, Cox Jr. LA, Keskinocak P, Kharoufeh JP, Smith JC (Eds.), John Wiley & Sons* 3: 1609–17.

Guimarães, Luis, Rui Santos, and Bernardo Almada-Lobo. 2011. "Scheduling Wafer Slicing by Multi-Wire Saw Manufacturing in Photovoltaic Industry: A Case Study." *The International Journal of Advanced Manufacturing Technology* 53: 1129–39 https://doi.org/10.1007/s00170-010-2906-x.

Hansen, Pierre, and Nenad Mladenovic. 2003. "A Tutorial on Variable Neighborhood Search." *Les Cahiers Du GERAD ISSN* 711: 2440.

Resende, Mauricio G C, and C Ribeiro. 1998. "Greedy Randomized Adaptive Search Procedures (GRASP)." *AT&T Labs Research Technical Report* 98 (1): 1–11.

Barhdadi, M., B. Benyacoub, and M. Ouzineb. "A Variable Neighborhood Search (Vns) Heuristic Algorithm Based Classifier for Credit Scoring." Paper presented at the International Conference on Advanced Intelligent Systems for Sustainable Development, 22-27 May 2022, Berlin, Germany, 2023.

Colombo, Fabio, Roberto Cordone, and Guglielmo Lulli. "A Variable Neighborhood Search Algorithm for the Multimode Set Covering Problem." *Journal of Global Optimization* 63 (2015): 461-80.

Geng, Lin, Luo Jinyan, Chen Xiang, Xu Haiping, and Xu Meiqin. "An Adaptive Feasible and Infeasible Search Algorithm for Solving the Set Covering Problem." Paper presented at the Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery.16th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD 2020), 19-21 Dec. 2020, Cham, Switzerland, 2021.

Hu, Enze, Jianjun He, and Shuai Shen. "A Hybrid Discrete State Transition Algorithm for Combinatorial Optimization Problems." *Frontiers in Energy Research* 11 (2023): 1148011.

Kalatzantonakis, P., A. Sifaleras, and N. Samaras. "A Reinforcement Learning-Variable Neighborhood Search Method for the Capacitated Vehicle Routing Problem." *Expert Systems With Applications* (/ 2023): 118812. https://doi.org/10.1016/j.eswa.2022.118812.

Klocker, B., and G. R. Raidl. "Solving a Weighted Set Covering Problem for Improving Algorithms for Cutting Stock Problems with Setup Costs by Solution Merging." Paper presented at the 16th International Conference on Computer Aided Systems Theory, EUROCAST 2017, 19-24 Feb. 2017, Cham, Switzerland, 2018.

Machado, André Manhães, Geraldo Regis Mauri, Maria Claudia Silva Boeres, and Rodrigo de Alvarenga Rosa. "A New Hybrid Matheuristic of Grasp and Vns Based on Constructive Heuristics, Set-Covering and Set-Partitioning Formulations Applied to the Capacitated Vehicle Routing Problem." *Expert Systems with Applications* 184 (2021/12/01/ 2021): 115556. https://doi.org/https://doi.org/10.1016/j.eswa.2021.115556..

Niu, D., X. Nie, L. Zhang, H. Zhang, and M. Yin. "A Greedy Randomized Adaptive Search Procedure (Grasp) for Minimum Weakly Connected Dominating Set Problem." *Expert Systems With Applications* (/ 2023): 119338. https://doi.org/10.1016/j.eswa.2022.119338.

Peng, Lu, Xiao Xiaoqiang, Wang Jijin, and Ning Weixun. "Improved Parallel Genetic Algorithm Based on Resetting Strategy and Hash Table." Paper presented at the 2019 4th International Seminar on Computer Technology, Mechanical and Electrical Engineering (ISCME 2019), 13-15 Dec. 2019, UK, 2020.

Radešček, Janez. "Exact and Approximate Uscp with Branch and Bound." *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (2021): 5-6. https://doi.org/10.1145/3449726.3463298.

Reyes, Victor, and Ignacio Araya. "A Grasp-Based Scheme for the Set Covering Problem." *Operational Research* 21, no. 4 (2021): 2391-408.

Souza Almeida, L., F. Goerlandt, R. Pelot, and K. Sorensen. "A Greedy Randomized Adaptive Search Procedure (Grasp) for the Multi-Vehicle Prize Collecting Arc Routing for Connectivity Problem." *Computers & Operations Research* 143 (/ 2022): 105804. https://doi.org/10.1016/j.cor.2022.105804.