An Efficient Approximate Greedy Algorithm to Solve the Minimum Set Cover Problem

Jorge Delgado, Héctor Ferrada and Cristóbal A. Navarro

Instituto de Informática, Universidad Austral de Chile, Chile

Abstract

The Minimum Set Cover Problem (MSCP) is a combinatorial optimization problem belonging to the NP-Hard class in computer science. For this reason, there is no algorithm that in the worst case ensures finding an optimal solution in polynomial time. For a given universe $\mathcal{X}$, the popular greedy heuristic, called `Greedy-SetCover`, is the main theoretical contribution to obtain an approximate solution for the MSCP in polynomial time, offering an optimal approximate ratio $(1 - o(|\mathcal{X}|)) \ln |\mathcal{X}|$. In this article, we propose an approximate algorithm for MSCP within a succinct representation of the input dataset, whose empirical performance improves `Greedy-SetCover` both in quality and execution time, while offering the same optimal approximation ratio for the problem. Our experiments show that the proposed algorithm is magnitudes of times faster than the aforementioned greedy one, obtaining on average a cardinality much closer to the optimal solution. Furthermore, because we work on a succinct representation that allows us to compute operations between sets using bitwise operators, we can process much larger datasets than state-of-the-art solutions. As a result, our proposal also is a suitable alternative for processing large datasets as required by the current Big Data era.

Keywords: Combinatorial optimization, Set cover, NP-Hard, Greedy algorithms, Approximate algorithms

1. Introduction

A *set-covering* for a given universe $\mathcal{X}$ is a group of subsets of $\mathcal{X}$ whose union includes all the elements of $\mathcal{X}$. Since there can be many different combinations of sets whose union gives the same universe, in some circumstances

Email address: `jorge.delgado01@alumnos.uach.cl`, `hferrada@inf.uach.cl`, `cristobal.navarro@uach.cl` (Jorge Delgado, Héctor Ferrada and Cristóbal A. Navarro)

it is desired to select only the best combination, or optimal, among all the available sets. When we define the optimum as the smallest number of sets to cover $\mathcal{X}$ and we require to determine what this optimal solution is, then we are talking about the Minimum Set Cover Problem (MSCP). The MSCP is an NP-hard optimization problem [17, 22] that we formally define as follows. Given an instance $(\mathcal{X}, \mathcal{F})$ as input dataset, where $\mathcal{F} = \{S_1, \ldots, S_m\}$ is the family of subsets and $\mathcal{X}$ is the universe formed by the union of all sets in $\mathcal{F}$, the problem consists in to find a set-covering $\mathcal{C} \subseteq \mathcal{F}$ of $\mathcal{X}$; i.e., $\mathcal{X} = \bigcup\limits_{S \in \mathcal{C}} S$ of minimum size. According to the definition it is possible that there is more than one solution for MSCP, in which case all of them are called optimal solutions for the given instance $(\mathcal{X}, \mathcal{F})$, and to solve MSCP, it is enough to find any of them. The Figure 1 shows an universe of twelve elements distributed in seven subsets $S_i \in \mathcal{F}, 1 \leq i \leq |\mathcal{F}|$ which are the input instance $(\mathcal{X}, \mathcal{F})$ for the problem, and for this input dataset the unique optimal solution is $\mathcal{C}^* = \{S_1, S_4, S_5\}$.

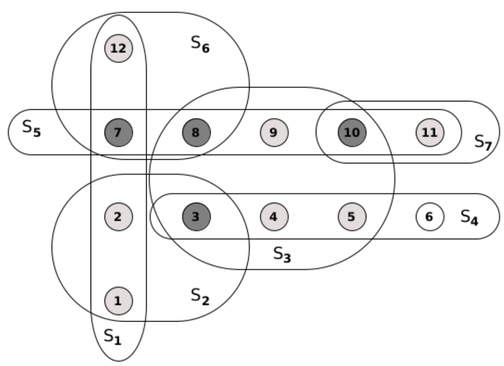

Figure 1: Illustration of an instance $(\mathcal{X}, \mathcal{F})$ of the MSCP, with $|\mathcal{X}| = 12$ elements and $\mathcal{F} = \{S_1, \ldots, S_7\}$. The optimal solution for this example is the minimum set-covering $\mathcal{C}^* = \{S_1, S_4, S_5\}$.

The MSCP is widely used in many optimization problems that require resources to be allocated. As a small example in combinatorics, within a multidisciplinary research project, we may need to select the minimum number of researchers to cover all the research areas and skills $\mathcal{X}$ necessary to carry out the project. Thus, a solution for MSCP it is enough. MSCP also appears naturally in tasks of coordination and distribution of finite resources within the industry. For instance, within industrial planning, in order to avoid overlaps and/or reduce operating costs, an optimal solution may be required to allocate worker shifts. In the same line of planning, the Transit Route Network Design Problem (TrNDP) can be derived from the MSCP [29] and use this solution to establish transport routes of a company. In another application, some model-based view planning methods also require a set-covering to solve the view planning problem (VPP), which involves determining the minimum number of views that must be used to reconstruct the part in 3D [32]. In industrial robot coordination, a solution for MSCP also is used to minimize synchronization points required by its algorithms [34]. A set-covering also appears as an optimization problem to make an efficient distribution of tasks in an *Internet of Things* (IoT) network within the context of limited energy [11]. In *Crew scheduling*, the problem of assigning a group of workers (a *crew*) to a set of tasks can be formulated from the MSCP[4]. Contardo et al., [16], propose a hybrid algorithm that iterates between computing a set-covering and constructing a Voronoi diagram to solve MSCP in geometric contexts. The MSCP has many other applications in different areas of research on optimization problems, such as industrial applications [13], information retrieval [35], in graph theory to determine a transversal in hypergraphs [14], in satellite systems for traffic allocation [33], among others.

In the worst case, any current method that determines the optimal combination of sets from $\mathcal{F}$, requires at least exponential time in the size of the family of sets, $|\mathcal{F}| = m$. We can homologate all the possible combinations from $\mathcal{F}$ of an exhaustive algorithm as if it were a binary counter of m bits, starting from 1 to 2^m. The binary counter determines each possible sets combination, where if the ith bit is one in a combination ρ, $1 \leq i \leq m$, then the set S_i is included in ρ. Thus, we need to evaluate each one of the 2^m states of the counter, checking if the combinations correspond to a true set-covering for $\mathcal{X}$, and remember the configuration with the fewest sets. In order to check if a group of sets includes all the elements of $\mathcal{X}$, it is necessary to perform a validation on all $n = |\mathcal{X}|$ elements, which requires $\Omega(n)$ time by using a sequential method that evaluates each element one by one. Therefore, the worst case time of an exhaustive optimal algorithm is $\Omega(n \cdot 2^m)$.

In this sense, there are many heuristics to improve, in some cases, the execution time of the exhaustive search that obtains the optimal solution to the problem. For example, we can narrow the search space by determining the smallest number of sets that any optimal solution must contain, as follows. If we find the smallest integer k such that the sum of the sizes of the k largest sets of $\mathcal{F}$ reaches the size of the universe, then we know that any optimal solution must contain at least k subsets. For that, we need to know what are the sizes of the k largest sets in $\mathcal{F}$, which we can obtain by ordering the family of subsets in $O(|\mathcal{F}| \lg |\mathcal{F}|)$ time. Thus, having found this k, we start the search by evaluating any combination of k sets; if we do not find a set-covering for the universe, then we iterate again increasing k by one unit and repeating the search for this updated k value. When we find a set-covering for $\mathcal{X}$, we stop the iteration reporting the last k value as the size of the optimal solution.

As we have seen, the MSCP has a great impact on academia, scientific research and several industry fields. However, under the assumption that it is very unlikely to find a polynomial-time algorithm to obtain the optimal set-covering, because its NP-completeness, great part of the effort of the scientific community is in the development of polynomial-time approximations for the problem. Meaning to offer faster solutions at the cost of a lower quality. Furthermore, we believe that in combination with efficient programming techniques, it is possible to build faster and/or smaller solutions to the problem, either to determine the optimal configuration or to design new approximate solutions.

The rest of this article is structured as follows, Section 2 details the related work in the field, the next Section 3 presents the new greedy solution for MSCP. Section 4 shows experimental results and finally Section 5 presents conclusions with future work.

2. Related Work

In 1972 Karp showed that the MSCP is NP-Hard [22], and therefore, very unlikely to be optimally solved in polynomial time. Two decades later, Lund and Yannakakis based on the work of Karp, showed in [27] that it is not possible to obtain a polynomial time approximate algorithm for solving the MSCP, with ratio better than $\frac{1}{4} \lg n$, unless NP has slightly superpolynomial time deterministic algorithms, that is $O(n^{poly \log n})$, where $n = |\mathcal{X}|$ is the universe size. Thus, every polynomial solution must have an approximate ratio $\geq \frac{1}{4} \log n$, showing that it is also difficult that a polynomial approximation method becomes $\leq \frac{1}{2} \log_2 n \simeq 0.72 \ln n$. A few years later, in 1998, Uriel Feige [18] adjusted the approximation ratio to $(1 - o(n)) \ln n$, but always considering that NP does not include slightly superpolynomial time algorithms. Therefore, given the NP-completeness of the MSCP which indicates that every optimal solution has superpolynomial time in the worst case, most of the current optimal strategies focus their success on the development of heuristics to prune the search space. The latter has the risk that for certain cases of the input instance it is not possible to do any pruning, or narrow the search only a little. For this reason, in the literature we find several optimal algorithms based on different pruning strategies for the search tree [2, 3, 5]. Among this type of algorithms, the method proposed by Lan et al. [25] finds the optimal set-covering by applying a search strategy called meta-heuristics for random priority search, introducing randomness into the resolution of the problem. Other exhaustive search algorithms are mainly based on branch-and-bound and branch-and-cut methods [1, 6, 19].

Maybe the major efforts in MSCP research are the design of new approximations, avoiding the restrictions imposed by the Hardness of the problem. In this line, there are several proposals in the state-of-the-art that try to obtain faster solutions and desirably close to the optimum quality. In 1973 Johnson published his Greedy algorithm [21] which determines an approximation for the MSCP in polynomial time. The algorithm 1, included in [17], describes the pseudocode of Johnson's method. His intuitive heuristic consists of iteratively selecting the set that covers the most number of remaining uncovered elements in the universe, subtracting the included items from the current universe and iterating until all items are covered. In 1979 Chvatal [15] investigated the work of Johnson to propose an approximate solution with very similar characteristics, which can be reduced to the Johnson's results under certain conditions. On the other hand, an interesting result appeared in [10], where the authors identified two issues that arise when determining a set-covering with metaheuristic approaches: solution infeasibility and set redundancy[1]. Accordingly, they propose a solution on a 1-flip neighbourhood structure to find a set-covering with metaheuristic by including a penalty approach to obtain an approximate solution to the problem. Lim et al., [26] also analyzed the main greedy algorithm approach to

[1]An incomplete set-covering is considered to be infeasible if one or more of the elements of the universe are uncovered (i.e., not really a solution). A set is considered to be redundant if all the elements covered by the set are also covered by other sets in the solution.

the MSCP by evaluating eager and lazy versions of the algorithm. Caprara et al.[12] proposed an heuristic based on Lagrangian relaxation to find an approximation for the problem, whose implementation won an important competition of that decade, called FASTER. Among randomized algorithms we highlight the method of Peuzin-Jubert et al., in [31], which introduces an heuristic called randomized rounding algorithm for MSCP, achieving an approximation ratio of $k(1-(c/n)^{1/k})$ for some small constant c.

Approximate minimum set-covering can also be treated with parallelized methods, for example, Koufogiannakis and Young [24] present parallel and distributed δ-approximation algorithms to cover problems, such as weighted set cover or weighted vertex cover, where δ is the maximum number of variables on which any constraint depends.

Gomes et al. [20] carried out an empirical study of approximation algorithms for the Vertex Cover and the Set Covering Problems. They tested the classical greedy algorithm against some variants of it, showing that, in general, the greedy strategy [21] is the fastest and the best approximation for the MSCP.

2.1. The Greedy Set Cover Algorithm

The solution with major acceptance in the literature is the greedy approximation introduced by Johnson in 1973 [21], which is also described in Cormen et al.[17]. This method is a polynomial time algorithm whose heuristic selects at each iteration the most promising subset among those available. The key idea is to select the set $S \in \mathcal{F}$ that contains the largest number of elements that have not yet been included by any set of the partial solution that has been built up to that iteration. The process ends the iterations when all the elements of the universe have been considered by at least one of the selected sets. This strategy ensures finding a solution $\mathcal{C}$ whose approximation relation is $\frac{|\mathcal{C}|}{|\mathcal{C}^*|} \leq \ln|\mathcal{X}| + 1$, where $\mathcal{C}^*$ is an optimal solution for the problem. The pseudocode described in 1 corresponds to the greedy set covering algorithm detailed in Cormen et al.[17] with $O(\min\{|\mathcal{X}|, |\mathcal{F}|\} \cdot |\mathcal{X}| \cdot |\mathcal{F}|)$ complexity time, in the worst case. This theoretical time is explained by the loop of lines 4...8, where it is necessary to determine the set S that maximizes $|\mathcal{U} \cap S|$. To do that, in each one of the $\min\{|\mathcal{X}|, |\mathcal{F}|\}$ iterations, we may need to examine up to $|\mathcal{F}|$ subsets of size $|\mathcal{X}|$, resulting in a bound of $O(|\mathcal{X}| \cdot |\mathcal{F}|)$ time.

Algorithm 1 GREEDY-SETCOVER algorithm in [17] to compute a set covering for the universe $\mathcal{X}$ and the family of subsets $\mathcal{F}$

```
Require: The universe 𝒳 and the family of subsets ℱ whose union is 𝒳
procedure GREEDY-SETCOVER(𝒳, ℱ)
    𝒰 = 𝒳
    𝒞 = ∅
    while 𝒰 ≠ ∅ do                                  ▷ O(min{|𝒳|, |ℱ|}) time
        select S ∈ ℱ that maximizes |𝒰 ∩ S|          ▷ O(|𝒰| · |ℱ|) time
        𝒰 = 𝒰 − S
        𝒞 = 𝒞 ∪ S
    end while
    return 𝒞
end procedure
```

The correctness of Algorithm 1 is ensured by the loop condition, which does not allow to finish the process until all the elements of the universe have been considered in the solution. The quality of the greedy algorithm has an approximation ratio of $O(\ln|\mathcal{X}|)$ [17], which is theoretically optimal considering the conditions for the problem discussed in the Introduction (Section 1).

3. An Approximate Greedy Set Cover Algorithm in a Succinct Data Representation

The classic greedy strategy described above achieves its time and quality by selecting the set with the largest number of uncovered elements by the partial solution. This polynomial-time algorithm 1 is essentially the best theoretical research result to date regarding to approximate solutions for the set cover problem. However, the method can take too long when the size of the data collection is large. Mainly because iteratively performs exhaustive searches for the best set among all available sets in $\mathcal{F}$, validating in the worst case up to $|\mathcal{F}|$ sets. Furthermore, the exaggerated generality of the algorithm invites the development of more adequate data structures to represent the sets; for instance, by using an environment that facilitates the computation of set operations such as Union and Intersection. These aspects have opened up a research opportunity that has given rise to this contribution.

Derived from Algorithm 1, we have designed a set cover approximated algorithm incorporating new heuristics to empirically improve the original greedy method. Thus achieving greater speedup and space savings, as well as guaranteeing optimal asymptotic complexity in terms of approximation quality —considering that this problem is encapsulated by the NP-hard class. We highlight two key points in our proposal: *i*- the design of an efficient and lightweight data structure that allows calculations of operations between sets directly in RAM; and *ii*- a new heuristic to prune the space search of the best set in each iteration. The final part of our proposal is an implementation of the new algorithm that improves both the execution time of the original method and the quality of the approximation.

When analyzing Algorithm 1, we can notice that the key step is in line 5, when it selects the next set $S \in \mathcal{F}$ to be included in the solution. This selection has to check all the remaining sets of $\mathcal{F}$, obligating to execute an exhaustive search in $\mathcal{F}$, requiring $O(|\mathcal{X}| \cdot |\mathcal{F}|)$ time in the worst case. The time of this selection can become very significant for large sizes of the universe $\mathcal{X}$ or the family of subsets $\mathcal{F}$. This is the most expensive task of each iteration in the algorithm, because most of these exhaustive searches are performed on a large subset of $\mathcal{F}$.

Consider Figure 1, where the optimal solution is $\{S_1, S_4, S_5\}$. The classic greedy algorithm starts by selecting the subset S_3 to build the solution $\mathcal{C} = \{S_3, S_1, S_4, S_7\}$, or including S_5 instead of S_7 in $\mathcal{C}$, whose size $|\mathcal{C}| = 4$. However, note that we can select S_4 as the first subset of the solution $\mathcal{C}$, because the element number 6, the rightmost in S_4, is not included by any other set. In this way, we can reduce the search space if we focus only on covering the elements that are present in the smallest number of subsets. Also hoping that, with this way of selecting each set $S \in \mathcal{F}$, other elements of the universe will also be covered, resulting in a possible faster and more efficient algorithm than 1 and also with a better approximation ratio. Thus, we iteratively select the set containing the largest number of uncovered elements that are present in the smallest number of subsets in $\mathcal{F}$. Formally, if an item $x \in \mathcal{X}$ is included exactly in k sets, then we call x as an *element of grade k*. In the example of the Figure 1, the white element, the item with identifier 6, has grade $k = 1$; the light gray elements (identifiers $1, 2, 4, 5, 9, 11, 12$) have grade $k = 2$; and the dark gray elements (identifier $3, 7, 8, 10$) have grade $k = 3$. We will then use this notion in order to provide a new greedy algorithm to compute the set-covering for the instance $(\mathcal{X}, \mathcal{F})$. Bellow we formally detail our algorithm, which pseudocode is described in 2.

Algorithm 2 SUCCINCT-SETCOVER algorithm to compute a set cover of the universe $\mathcal{X}$ and the family of subsets $\mathcal{F}$.

Require: The universe of elements $\mathcal{X}$ and the family of subsets $\mathcal{F}$ whose union is $\mathcal{X}$

procedure SUCCINCT-SETCOVER($\mathcal{X}, \mathcal{F}$)
    be MAP a map data structure of length $|\mathcal{X}|$ to store, for each $x_i \in \mathcal{X}$, the lists of ID-subsets where x_i is.
    MAP $\leftarrow$ CREATEMAP($\mathcal{X}, \mathcal{F}$) ▷ $O(|\mathcal{X}| \cdot |\mathcal{F}|)$ time
    SORTUNIVERSE(MAP, $\mathcal{F}$) ▷ $O(|\mathcal{X}| \log |\mathcal{X}|)$ time
    $\mathcal{C} = \varnothing$
    $\mathcal{U} = \mathcal{X}$
    $ini = k = 1$
    while $\mathcal{U} \neq \varnothing$ **do** ▷ $O(\min\{|\mathcal{X}|, |\mathcal{F}|\})$ time
        $end \leftarrow$ FINDRANGE(MAP, ini, k) ▷ $O(|\mathcal{X}|)$ time for all the calls
        $\ell = end - ini + 1$ ▷ numbers of elements of grade k to cover
        while $\ell > 0$ **do**
            $\langle S, \delta \rangle \leftarrow$ FINDSUBSET(MAP, ini, end) ▷ $O(\frac{|\mathcal{X}|}{W}|\mathcal{F}|)$ time
            $\mathcal{U} = \mathcal{U} - S$ ▷ $O(\frac{|\mathcal{X}|}{W})$ time
            $\mathcal{C} = \mathcal{C} \cup S$ ▷ $O(\frac{|\mathcal{X}|}{W})$ time
            $\ell = \ell - \delta$
        end while
        $ini = end + 1$
        $k = k + 1$
    end while
    return $\mathcal{C}$
end procedure

We first detail the function of each procedure included in the pseudocode of the algorithm.

- CREATEMAP($\mathcal{X}, \mathcal{F}$), procedure that creates and returns a map data structure, called MAP, of pairs $(x_i, L_i[\,])$, where for each $x_i \in \mathcal{X}$, $L_i[\,]$ is the list of all subsets $S_i \in \mathcal{F}$ in which x_i is. It can be built in $O(|\mathcal{X}| \cdot |\mathcal{F}|)$ time by simple iterative inspection.

- SORTUNIVERSE(MAP, $\mathcal{F}$), procedure that sorts the MAP structure of size $|\mathcal{X}|$ in increasing order with respect to the number of times that each $x_i \in \mathcal{X}$ appears in the subsets of $\mathcal{F}$. In this process, for each x_i, with the same MAP, we obtain in constant time the length of each list of subsets that contain x_i. Thus, MAP is sorted in $O(|\mathcal{X}| \log |\mathcal{X}|)$ time.

- $\textsc{findRange}(\textsc{map}, ini, k)$, procedure that determines and returns the right limit end of the range $\textsc{map}[ini \ldots end]$ of all elements $x \in \mathcal{X}$ of grade k. Thanks that MAP is sorted, determining each interval $I = \textsc{map}[ini \ldots end]$ can be done in linear time over the size $|I|$. If there are no elements of grade k, then returns $ini - 1$.

- $\textsc{findSubSet}(\textsc{map}, ini, end)$, procedure that determines and returns the pair $\langle S, \delta \rangle$, where $S \in \mathcal{F}$ is the subset that contains the major number δ of minimal uncovered elements inside the range $\textsc{map}[ini \ldots end]$ of all elements with the same grade. If there are ties, then select the subset that maximizes $|S \bigcap \mathcal{U}|$ among all the lists $\textsc{map}[x_i].list$, $\forall\ x_i \in \textsc{map}[ini \ldots end]$.

As input, we are given the instance $(\mathcal{X}, \mathcal{F})$, which are the universe of elements $\mathcal{X}$ and the family of subsets $\mathcal{F}$, whose union is $\mathcal{X}$. We also require to build a MAP data structure[2] to associate to each $x \in \mathcal{X}$ the list of all subsets $S \in \mathcal{F}$ where x is —created at the line 3 in the pseudocode. At line 4, we sort the universe $\mathcal{X}$ in MAP (by the length of the list that contains each element) by calling to the procedure SORTUNIVERSE. We build the solution $\mathcal{C}$ incrementally, which is initially an empty set –see line 5. We start iterating with $\mathcal{U} = \mathcal{X}$ while $\mathcal{U}$ has elements, which means that the size of our solution $\mathcal{C}$ is lower than $|\mathcal{X}|$ —line 8. The process finds, at line 9, each range $\textsc{map}[ini \ldots end]$ of all elements $x \in \mathcal{X}$ of grade k, starting with $k = ini = 1$. These elements of grade k, are also called *Minimal Elements*, because we look for uncovered elements of minimal grade k among all the remaining elements not yet included by the partial solution $\mathcal{C}$. Therefore, at the beginning, we start searching in MAP the interval of all elements that are included only by $k = 1$ subsets of $\mathcal{F}$; that is, in the first group of iterations (the inner loop at lines 11—16) the minimal elements are items of degree $k = 1$. Thus, in the next iterations of the while-cycle of lines 8–19 we search for the next interval of minimal items of grade $k = 2$. As in the MAP structure, the ranges of elements are grouped by their degree value, placing one after the other as the degree of the elements increases. Therefore, we now loop in the inner while-cycle, lines 11—16, while the range size ℓ be greater than 0. It means

[2] This corresponds to a BST or a Red/Black Tree [17, 23], where each key has associate an additional data structure such as `std::map`, which is provided by the GNU C++ Standard Library and implements a Red/Black Tree.

that there are uncovered elements of degree k and, consequently, we must continue including subsets in the solution until we cover all the minimal elements of the range.

The correctness of our algorithm is given by the condition of the outer loop of line 8, which does not end as long as there is at least one uncovered element.

With the example of Figure 1, unlike the classic greedy algorithm that starts by searching exhaustively among all the subsets in $\mathcal{F}$ to select the subset S_3, the first search of the SUCCINCT-SETCOVER algorithm is only among the sets that contains at least one element of minimal degree, since we start with $k = 1$, the only set that satisfies the condition is S_4. The next selected set, for $k = 2$, is S_1, which once included elements of degree 2 remain uncovered, so S_5 is chosen in the second iteration of the inner loop for $k = 2$. Note that in the example we never search for elements of degree $k = 3$ because these were implicitly included when we added the sets to cover the elements of degree $k = 1$ and $k = 2$.

3.1. *Asymptotic Analysis of the* SUCCINCT-SETCOVER *Algorithm*

We detail the asymptotic analysis of our SUCCINCT-SETCOVER Algorithm 2 for both, the time consumption and the quality of the solution.

To obtain a threshold of the time, let us break down the pseudocode of the algorithm. At the beginning we need to build a sorted MAP dictionary, requiring a time complexity of $O(|\mathcal{X}| \cdot |\mathcal{F}| + |\mathcal{X}| \log |\mathcal{X}|) = O(|\mathcal{X}| \cdot |\mathcal{F}|)$, as described above. Later, at line 9, all the calls for the procedure FINDRANGE require $O(|\mathcal{X}|)$ time, which is achieved by taking advantage of the order in MAP, performing a simple scan. At line 12, each call of the procedure FINDSUBSET(MAP, ini, end) requires a time of $len \cdot O(\frac{|\mathcal{X}|}{W}|\mathcal{F}|)$, with $len = end - ini + 1$, at the worst case with $len \leq \min\{|\mathcal{X}|, |\mathcal{F}|\}$. The explanation is given that we can solve any set operations between a pair of sets —such as UNION, INTERSECTION and DIFFERENCE— in linear time on the size $|\mathcal{X}|/W$, where W is the memory word size in the CPU architecture[3]. This is achieved with efficient computing techniques that are detailed in Section 3.2. Accordingly, we also take $O(|\mathcal{X}|/W)$ time for computing $\mathcal{U} - S$ and

[3]Note that if $n' = |\mathcal{X}|/W$ is small, then the operation is made in constant time.

$\mathcal{C} \cup S$, at lines 13—14. Altogether, the total run time of our algorithm is $O(\min\{|\mathcal{X}|, |\mathcal{F}|\}\frac{|\mathcal{X}|\cdot|\mathcal{F}|}{W})$ at the worst case, which is clearly polynomial.

Let us now continue with the detail of the approximation relation of our solution, which we will proof that it cannot be greater than $|\mathcal{C}^*|(\ln|\mathcal{X}|+1)$ in the worst case, where $\mathcal{C}^*$ is an optimal solution for the instance. To perform the proof, we will use the following inequality that relates the Harmonic Number $H(n)$ with the natural logarithm [17, 28].

$$H(n) = \frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k} \leq \ln n + 1 \tag{1}$$

Let S_i denote the ith subset selected by our SUCCINCT-SETCOVER Algorithm 2 and k_i the current grade of minimal items at the ith iteration respect to the inner loop (lines 11—16). Thus, the ith iteration corresponds when the procedure findSubSet(...) selects the best subset $S_i \in \mathcal{F}$ from the range of degree k_i, which is the one that contains the largest number of uncovered minimal elements (i.e., elements of degree k_i). We also establish that each selected subset S_i contributes with a cost of 1 to $\mathcal{C}$, cost that is distributed by all the new minimal items $x \in S_i$, and the sum of all these costs is also the cardinality of our solution $\mathcal{C}$. We define the cost of $x \in S_i$ as follow

$$c_x = \begin{cases} \frac{1}{\textsc{minimal}(S_i - \mathcal{C}_{i-1})} & \text{, if } x \in S_i \text{ has grade } k_i \text{ and } x \notin \mathcal{C}_{i-1} \\ 0 & \text{, otherwise} \end{cases} \tag{2}$$

where $\mathcal{C}_{i-1} = S_1 \cup S_2 \cup \ldots \cup S_{i-1}$ is the partial solution built up to $(i-1)$th iteration and $\textsc{minimal}(S_i - \mathcal{C}_{i-1})$ corresponds to the number of items of grade k_i added by first time by the algorithm in the ith iteration. Seen another way, if S_i is selected, then $\textsc{minimal}(S_i - \mathcal{C}_{i-1})$ is the number of new minimal elements that S_i contributes to the solution. Thus, we have

$$\sum_{x \in S_i} c_x = 1 \tag{3}$$

Now, for any set S belonging to the family $\mathcal{F}$ and by using our definition of cost the following relation is always true

$$\sum_{x \in S} c_x \leq H(|S|) \tag{4}$$

In order to prove (4), consider any set $S \in \mathcal{F}$ and any $i = 1, 2, \ldots, |\mathcal{C}|$, and let $u_i = \textsc{minimal}(S - (S_1 \cup S_2 \cup \ldots \cup S_i))$, as the number of minimal elements in S that remain uncovered after the algorithm has selected sets $S_1, S_2, \ldots, S_i$. Let $k' \geq 1$ the minimal grade for any element in the universe and u_0 $=\textsc{minimal}(S)$ as the number of elements with grade k' in S. Let $t \leq |\mathcal{C}|$, the smallest index such that $S \subseteq (S_1 \cup \ldots \cup S_{t-1} \cup S_t)$ but $\exists\ x \in S : x \notin (S_1 \cup \ldots \cup S_{t-1})$. Thus, $u_t = 0$ and $u_j \leq u_{j+1}$, $\forall\ j = 0, 1, \ldots, t-1$. Observe that $(u_{i-1} - u_i) \geq 0$ minimal elements of S are covered for the first time by S_i, for $i = 1, 2, \ldots, t$. Then we have (for S)

$$\begin{aligned}
\sum_{x \in S} c_x &= \sum_{i=1}^{t} (u_{i-1} - u_i) \cdot c_x \\
&\leq \sum_{i=1}^{t} (u_{i-1} - u_i) \frac{1}{\textsc{minimal}(S_i - (S_1 \cup S_2 \cup \ldots \cup S_{i-1}))}
\end{aligned}$$

because it is possible that for some elements in S the cost is 0. We can also see that $\textsc{minimal}(S_i - (S_1 \cup S_2 \cup \ldots \cup S_{i-1})) \geq \textsc{minimal}(S - (S_1 \cup S_2 \cup \ldots \cup S_{i-1}))$, since the algorithm selects the set S_i as the set with the largest number of uncovered elements of grade k_i (i.e. $\textsc{minimal}(S_i) \geq \textsc{minimal}(S), \forall\ S \in \mathcal{F}$ not included yet). Then, we now have

$$\begin{aligned}
\sum_{x \in S} c_x &\leq \sum_{i=1}^{t} (u_{i-1} - u_i) \frac{1}{\textsc{minimal}(S - (S_1 \cup S_2 \cup \ldots \cup S_{i-1}))} \\
&\leq \sum_{i=1}^{t} (u_{i-1} - u_i) \frac{1}{u_{i-1}} \\
&= \sum_{i=1}^{t} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{u_{i-1}}
\end{aligned}$$

giving that $u_{i-1} \geq j$, we have

$$\begin{aligned}
\sum_{x \in S} c_x &\leq \sum_{i=1}^{t} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{j} \\
&= \sum_{i=1}^{t} \left(\sum_{j=1}^{u_{i-1}} \frac{1}{j} - \sum_{j=1}^{u_i} \frac{1}{j} \right) \\
&= \sum_{i=1}^{t} \left(H(u_{i-1}) - H(u_i) \right) \\
&= H(u_0) - H(u_t) \\
&= H(\textsc{minimal}(S)) - 0 \qquad (\textit{because } u_0 = \textsc{minimal}(S)) \\
&\leq H(|S|) \qquad\qquad\qquad (\textit{because } \textsc{minimal}(S) \leq |S|)
\end{aligned}$$

We now can prove Eq. (1). Thus, giving our definition of cost, we have

$$|\mathcal{C}| = \sum_{x \in \mathcal{X}} c_x \tag{5}$$

As each element $x \in \mathcal{X}$ is at least in one set of the optimal solution $\mathcal{C}^*$, then we have

$$|\mathcal{C}| = \sum_{x \in \mathcal{X}} c_x \leq \sum_{S \in \mathcal{F}} \sum_{x \in \mathcal{S}} c_x \tag{6}$$

For (4) we have

$$\begin{aligned}
|\mathcal{C}| &\leq \sum_{S \in \mathcal{F}} |H(S)| \\
&\leq |\mathcal{C}^*| \cdot H(\max\{|S| : S \in \mathcal{F}\})
\end{aligned}$$

Finally, by using (1) we conclude that

$$|\mathcal{C}| \leq |\mathcal{C}^*|(\ln |\mathcal{X}| + 1)$$

which completes the proof that the approximation ratio of our algorithm is $\ln |\mathcal{X}| + 1$.

3.2. Efficient Computing Techniques in Succinct Space

In order to achieve sublinear time for operations on sets such as UNION, INTERSECTION and DIFFERENCE, we first represent the input datasets $(\mathcal{X}, \mathcal{F})$ as a bit matrix $M_{|\mathcal{F}|\times|\mathcal{X}|}$ of $|\mathcal{F}| \cdot |\mathcal{X}|$ bits. Thus, the ith row $M[i]$ in $M_{|\mathcal{F}|\times|\mathcal{X}|}$ corresponds to the representation of the ith subset S_i, and the jth element of the universe $\mathcal{X}$ is represented by the jth bit in each row of the matrix. We then set $M[i][j] = 1$ if the jth element is present in the subset S_i and with 0 if this element does not belong to the set. We now explain $\forall\ i, j, 1 \leq i \leq |\mathcal{X}|, 1 \leq j \leq |\mathcal{F}|$, how to compute $S_i \cup S_j$, $S_i \cap S_j$ or $S_i - S_j$ in constant time from the bit representation.

First, consider the singular case when $|\mathcal{X}| \leq W$, where W is the memory word size of the CPU architecture. Let p and q be two words of memory, of W bits each, to represent the $\leq W$ elements of the sets S_i and S_j respectively. Therefore, we can solve any of these set operations on p and q simply by applying the *bitwise* operators AND, OR and NEGATION. The table 1 shows how to compute these operations between sets by using bitwise operator.

set operation	bitwise operation
$S_i \cup S_j$	p OR q
$S_i \cap S_j$	p AND q
$S_i - S_j$	p AND $\neg q$

Table 1: The equivalent bitwise operations for set operations. The sets S_i and S_j are represented by the bit sequences p and q respectively, whose lengths are at most $|W|$ bits (i.e., $|\mathcal{X}| \leq W$ elements). The symbol $\neg$ corresponds to the negation operator (i.e., $\neg 1 = 0$ and $\neg 0 = 1$). Therefore, to compute any set operation can be done in $O(1)$ by using bitwise operator on p and q.

For universes of any length, $|\mathcal{X}| > W$, we can process the $|\mathcal{X}|$ bits by implement a loop routine in $O(|\mathcal{X}|/W)$ time; where p and q are bit arrays of $|\mathcal{X}|/W$ words of memory.

4. Experimental Results

In order to evaluate the empiric performance of the algorithms, we provide three implementations in C++[4]. The first, called `Exhaustive-SC`, finds the optimal set-covering by means of exhaustive searches. This is used as a baseline for the smaller datasets described in the experiment 2. The second, referred to as `Greedy-SC`, is the implementation of the classical approximate greedy set cover solution given by the Algorithm 1. The last implementation corresponds to our SUCCINCT-SETCOVER algorithm which is referred to as `Succinct-SC`. We test the implementations by using different instances $(\mathcal{X}, \mathcal{F})$ of different lengths and different density for each set. All runtimes obtained are shown in logarithmic scale.

4.1. Setup

The experiments mentioned were performed in the Patagón Supercomputer [30], using a compute node with 2× AMD EPYC 7742 CPU (2.6Ghz, 64-cores, 256MB L3 Cache), equipped with 1TB RAM DDR4-3200Hz. Only a single thread of execution was used. The OS was Linux (Ubuntu 20.04, 64bit) running kernel 5.8.0. All programs were compiled using `g++` version 9.3.0. The runtime corresponds to real (wallclock) time from the `Chrono` library, using the `high_resolution` clock method.

4.2. Experiments with Random-Generated Datasets

In the first cycle of experiments, Figure 2, we compute the exact solution for the MSCP and the approximate responses to each instance. Since the time complexity of the optimal solution is superpolynomial, we only run experiments for instances small enough to get the answer in a reasonable amount of time. The experiment involves 4 cases with different sizes of $|\mathcal{F}|$, and each one with 5 instances with an expected universe of $|\mathcal{X}| = 70$, randomly generated using a Gaussian distribution, whose arithmetic mean was obtained from a uniform distribution over the range $\left[0..|\mathcal{X}|\right]$ and the standard deviation is $0.125|\mathcal{X}|$.

The dark gray line gave by the `Exhaustive-SC` corresponds to the minimum cardinality to the problem and therefore, the lower bound for any approximation to the problem.

[4]The repository with the code is available at `https://github.com/kokee07/succinct_mscp`

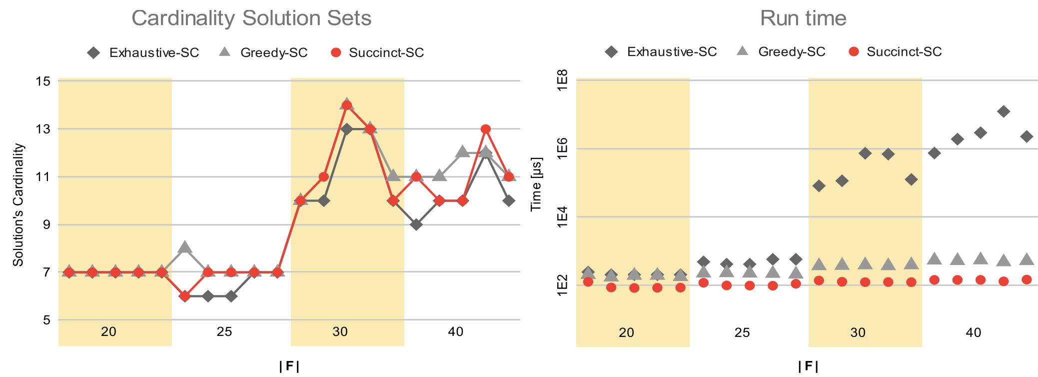

Figure 2: Cardinality (left) and running time (right) of our `Succinct-SC` approximation compared to the exact solution, `Exhaustive-SC`, and the classical greedy heuristic, `Greedy-SC`, for low size random generated datasets.

We can appreciate how, on average, the `Succinct-SC` is faster (right) and smaller (left) than the `Greedy-SC` approximation.
Given the exponential growth in time required for the `Exhaustive-SC` to be computed on large datasets, the following experiments will only consider the approximate solutions. The second cycle of experiments, Figures 3 to 5, involves 9 random-generated cases, using the same data generator mentioned before, but increasing the value of $|\mathcal{X}|$ and $|\mathcal{F}|$ progressively. Each figure includes 3 cases with 5 instances each, grouped by the size of $|\mathcal{F}|$.

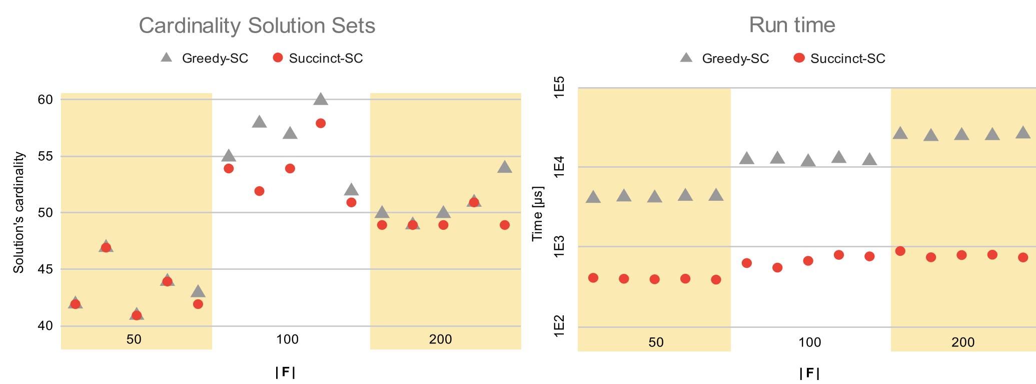

Figure 3: Cardinality and running time of our Succinct-Set Cover algorithm compared to the Greedy Set Cover for random-generated datasets with values: $|\mathcal{X}| \in [200..300]$ and $|\mathcal{F}| \in [50, 100, 200]$.

Figure 3 (left) shows an irregular difference between the quality of the solutions of both implementations, being `Succinct-SC` at least as good as `Greedy-SC`. Regarding time (right), as $|\mathcal{F}|$ increases, the distance between the execution time of each algorithm keeps increasing, at least by an order of magnitude.

Figure 4 shows the results over a medium size dataset. Achieving a more regular distance between the solutions quality, being our algorithm sightly better than the original Greedy approach. At the right side, each case of $|\mathcal{F}|$ size has an increase in the order of magnitude between the both solutions, evidencing the higher complexity of operations required for `Greedy-SC`.

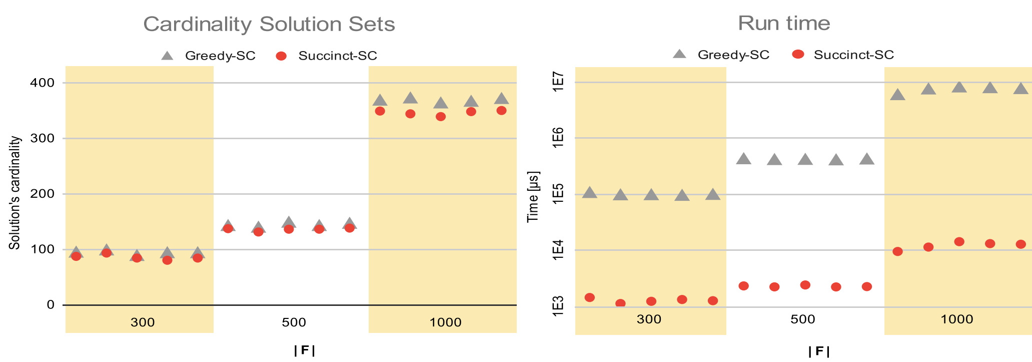

Figure 4: Cardinality and running time of our Succinct-Set Cover algorithm compared to the Greedy Set Cover for random-generated datasets with values: $|\mathcal{X}_e| \in [500..2.000]$ and $|\mathcal{F}| \in [300, 500, 1.000]$.

In the Figure 5, we generate higher order of $|\mathcal{F}|$. The left side shows the cardinality of the solution sets in thousands of subsets, in which our `Succinct-SC` algorithm computes a lower, and hence better, solution for each instance than the `Greedy-SC`. At the right side, the distance in order of magnitude keeps the same behaviour as the previous experiments.

The next cycle of experiments measures the advantage of our algorithm for the case where there are elements that are included by a single subset of $\mathcal{F}$, or by very few sets of $\mathcal{F}$. In this scenario, our algorithm takes advantage by performing, in the first iteration cycles, a much more efficient pruning of the search space than the traditional greedy approach. To emulate that scenario, we implement a new generator to secure the existence of elements contained only by one subset of $|\mathcal{F}|$. To achieve this, the generator takes

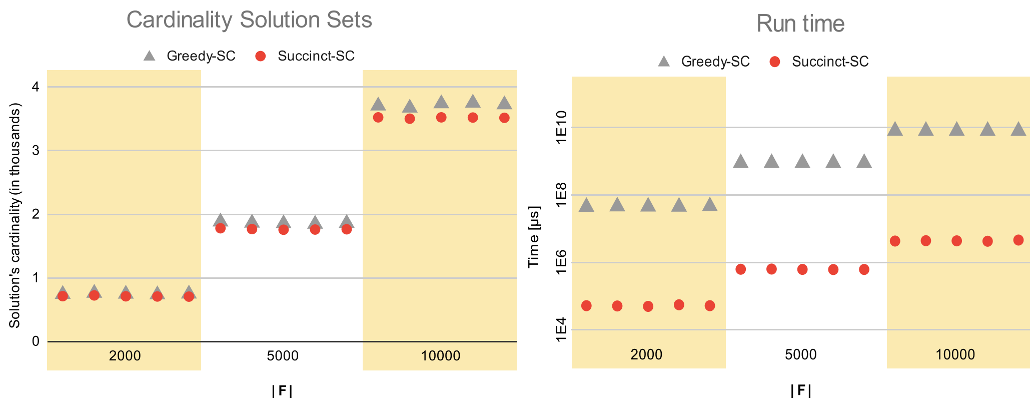

Figure 5: Cardinality and running time of our Succinct-Set Cover algorithm compared to the Greedy Set Cover for random-generated datasets with values: $|\mathcal{X}| \in [4.000..20.000]$ and $|\mathcal{F}| \in [2.000, 5.000, 10.000]$.

as input the expected universe size $|\mathcal{X}|$, the total number of subsets $|\mathcal{F}|$, a number $\mathcal{T}$ of samples to generate and a parameter p. Thus, we ensure that the first $p * |\mathcal{X}|$ elements of the universe will be randomly assigned to one set each; and then we generate all the rest of the $\mathcal{T}$ samples by using a uniform distribution over the range $[p * |\mathcal{X}| \ldots |\mathcal{X}|]$.

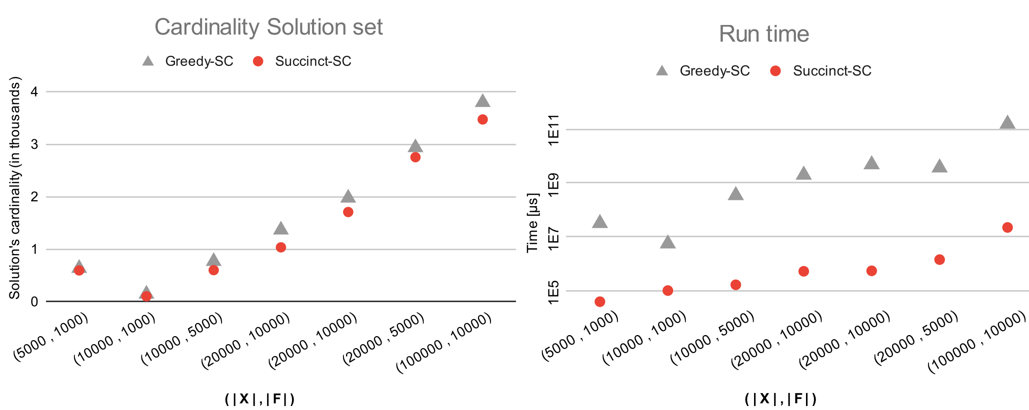

Figure 6: Cardinality and running time of our Succinct-Set Cover algorithm compared to the Greedy Set Cover for optimal random-generated datasets.

Figure 6 shows the results obtained from 7 different instances of optimal datasets, with the pair values ($|\mathcal{X}|$,$|\mathcal{F}|$) of each case indicated in the X-axis. The results show that `Succinct-SC` obtains a better solution for any instance within this scenario, with a greater difference than the previous cases. Furthermore, as shown on the right side of Figure 6, in these cases a difference of up to 4 orders of magnitude is reached between the times of both algorithms.

4.3. Experiments with Natural/Real Datasets

The following experiments show the results of Dataset benchmarks from OR-Library[9], including the SET PARTITIONING PROBLEMS datasets from K. L. Hoffman and D. Levine[8], used in genetic algorithms, in Figure 7 and SET COVER PROBLEM datasets from a Railway System, contributed by Paolo Nobili[7] in Figure 8.

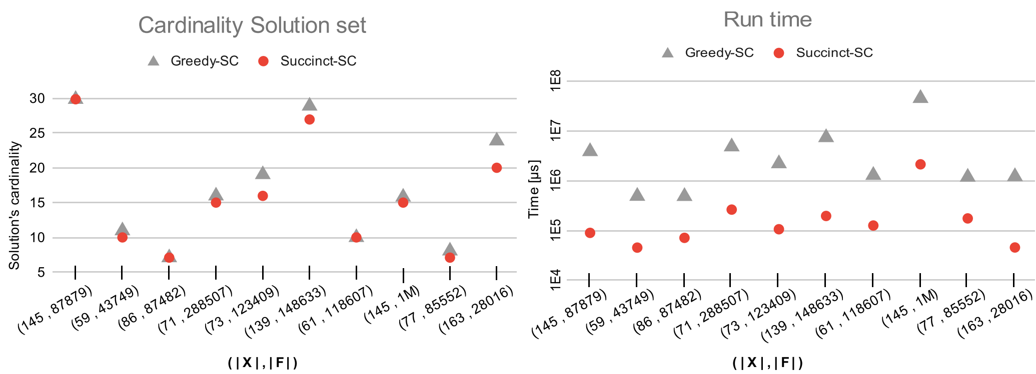

Figure 7: Cardinality and running time of our SUCCINCT-SET COVER algorithm compared to the GREEDY SET COVER for Genetic Datasets with a low $\mathcal{F}$ cardinality.

The decision of using the genetic algorithm Dataset as a Natural case is supported due to the low universe cardinality with a high number of combinations from $\mathcal{F}$ presented. For our purpose, the constraints associated to SET PARTITIONING PROBLEMS in the dataset are ignored, such as that any element must be included in one and only one set in the solution. In Figure 7 we can see, at the left, a more erratic behaviour from each case due to their natural origin. At the right, our SUCCINCT-SET COVER is at least 1 order of magnitude faster than the greedy approach.

As the last experiment, we test the SET COVER PROBLEM solution for a larger dataset from Italian railways [7], where a set cover solution represents a selection of routes that cover all the stations. The Figure 8 shows the results from 7 cases, indicating their respective $|\mathcal{X}|$ and $|\mathcal{F}|$ at the X-axis.

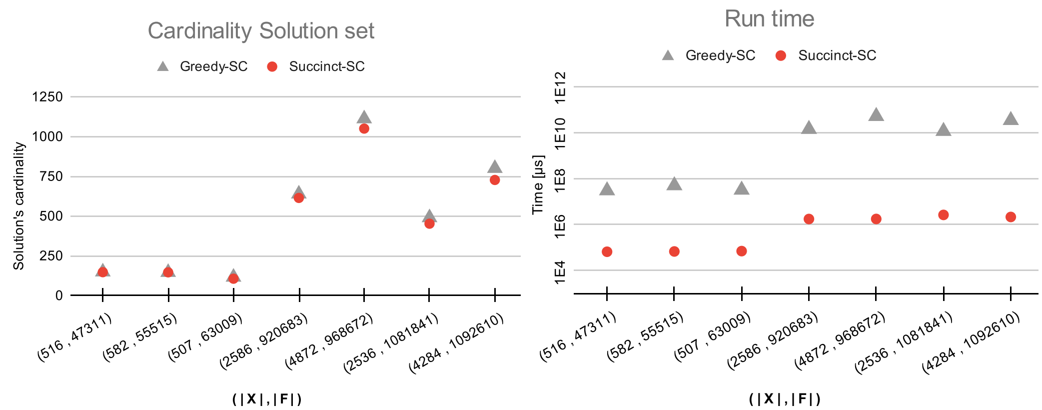

Figure 8: Cardinality and running time of our Succinct-Set Cover algorithm compared to the Greedy Set Cover for Railways Datasets.

The results shown in Figure 8 are consistent with the all the previous experiments, with our Succinct-Set Cover algorithm achieving a solution at least as good as Greedy Set Cover for those cases with a low universe, and a better solution for the other cases with a high universe of elements. At the right side, the difference in order of magnitude between both algorithm varies from 2 to 3 orders in each case. This is highly important since our technique could find a better solution in minutes for any real case, in contrast with the hours or days of processing attributed to the Greedy algorithm.

5. Conclusions and Future Work

We have presented a polynomial time approximate solution to solve the MSCP, offering an innovative algorithm that includes data compression techniques to speed up the process. The proposed `Succinct-SC` algorithm preprocesses the input dataset to create a succinct representation of the sets of the family $\mathcal{F}$; which also allows us to calculate operations between sets in a much more efficient way than with the traditional approach. Thanks to this, we improve the execution time of the `Greedy-SC` algorithm significantly, both in theory and in practice. This theoretical asymptotic bound is an important part of the contributions of this work. The algorithm `Succinct-SC` also offers a theoretically optimal ratio approximation under the assumption that there is no polynomial time algorithm that solves any of the problems belonging to the NP-Hard class. We have implemented our algorithm to measure the computation time and the quality of the approximation, demonstrating that

`Succinct-SC` improves the query time by orders of magnitude with respecto to the traditional greedy approximation, `Greedy-SC` algorithm, while achieving a better approximation rate. In summary, the results obtained with our algorithm are very favorable for the datasets tested, which included randomly generated ones as well as real ones from the state of the art.

One aspect to highlight is in relation to the efficiency achieved thanks to the succinct representation of the data, which allows computing operations between sets at high speed, always in a succinct space. We establish that the main focus when using this technique is to achieve greater speed and not necessarily compression. Since we are working on an approximation, which first has to guarantee a good quality response —we achieve this with a theoretical proof of the bound of the algorithm— but at high speed —we achieve it by providing an implementation of an algorithm polynomial that also, in practice, solves bitwise set operations.

The succinct/compressed representation of the data is a relevant subject for future work, where the goal will be to design an optimal algorithm that works in a succinct space, also incorporating CPU/GPU parallelism. In terms of compression and based on the tested scenarios, we expect that our implementation can reach volumes that are difficult for other implementations to reach; standing out even more in universes with a large number of subsets (when $\mathcal{F}$ is very large), where each subset has a large number of elements on average.

6. Acknowledgment

This research was supported by the Fondecyt grant #11221029 and by the Patagón supercomputer of Universidad Austral de Chile (FONDEQUIP EQM180042).

References

[1] Balas, E. and Carrera, M. C. (1996). A dynamic subgradient-based branch-and-bound procedure for set covering. *Oper. Res.*, 44(6):875–890.

[2] Balas, E. and Ho, A. (1980). *Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study*, pages 37–60. Springer Berlin Heidelberg, Berlin, Heidelberg.

[3] Balas, E. and Ho, A. (2009). *Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study*, volume 12, pages 37–60.

[4] Barnhart, C., Cohn, A. M., Johnson, E. L., Klabjan, D., Nemhauser, G. L., and Vance, P. H. (2003). *Airline Crew Scheduling*, pages 517–560. Springer US, Boston, MA.

[5] Beasley, J. (1987). An algorithm for set covering problem. *European Journal of Operational Research*, 31(1):85–93.

[6] Beasley, J. and Jørnsten, K. (1992). Enhancing an algorithm for set covering problems. *European Journal of Operational Research*, 58(2):293–300. Practical Combinatorial Optimization.

[7] Beasley, J. E. Or-library, set cover problems dataset. [dataset].

[8] Beasley, J. E. Or-library, set partitioning problems dataset. [dataset].

[9] Beasley, J. E. (1990). Or-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072.

[10] Bilal, N., Galinier, P., and Guibault, F. (2013). A new formulation of the set covering problem for metaheuristic approaches. *ISRN Operations Research*, 2013:10 pages.

[11] Burhan, H. M., Attea, B. A., Abbood, A. D., Abbas, M. N., and Al-Ani, M. (2021). Evolutionary multi-objective set cover problem for task allocation in the internet of things. *Applied Soft Computing*, 102:107097.

[12] Caprara, A., Fischetti, M., and Toth, P. (1999). A heuristic method for the set covering problem. *Operations Research*, 47(5):730–743.

[13] Chan, T. M., Grant, E., Könemann, J., and Sharpe, M. (2012). Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, page 1576–1585, USA. Society for Industrial and Applied Mathematics.

[14] Chen, J., Mi, J., and Lin, Y. (2020). A graph approach for fuzzy-rough feature selection. *Fuzzy Sets and Systems*, 391:96–116. Computer Science.

[15] Chvatal, V. (1979). A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4(3):233–235.

[16] Contardo, C. and Hertz, A. (2021). An exact algorithm for a class of geometric set-cover problems. *Discrete Applied Mathematics*, 300:25–35.

[17] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, 3rd Edition*. MIT Press.

[18] Feige, U. (1998). A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652.

[19] Fisher, M. L. and Kedia, P. (1990). Optimal solution of set covering/partitioning problems using dual heuristics. *Manage. Sci.*, 36(6):674–688.

[20] Gomes, F. C., Meneses, C. N., Pardalos, P. M., and Viana, G. V. R. (2006). Experimental analysis of approximation algorithms for the vertex cover and set covering problems. *Computers & Operations Research*, 33(12):3520–3534. Part Special Issue: Recent Algorithmic Advances for Arc Routing Problems.

[21] Johnson, D. S. (1973). Approximation algorithms for combinatorial problems. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, page 38–49, New York, NY, USA. Association for Computing Machinery.

[22] Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA.

[23] Knuth, D. E. (1973). *The Art of Computer Programming. Vol.3: Sorting and Searching.*

[24] Koufogiannakis, C. and Young, N. E. (2009). Distributed and parallel algorithms for weighted vertex cover and other covering problems. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*, PODC '09, page 171–179, New York, NY, USA. Association for Computing Machinery.

[25] Lan, G., DePuy, G. W., and Whitehouse, G. E. (2007). An effective and simple heuristic for the set covering problem. *European Journal of Operational Research*, 176(3):1387–1403.

[26] Lim, C. L., Moffat, A., and Wirth, A. (2014). Lazy and eager approaches for the set cover problem. In *Proceedings of the Thirty-Seventh Australasian Computer Science Conference - Volume 147*, ACSC '14, page 19–27, AUS. Australian Computer Society, Inc.

[27] Lund, C. and Yannakakis, M. (1994). On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981.

[28] Olaikhan, A. (2021). *An Introduction to the Harmonic Series and Logarithmic Integrals: For High School Students Up to Researchers*. Amazon Digital Services LLC - KDP Print US.

[29] Owais, M., Osman, M. K., and Moussa, G. (2016). Multi-objective transit route network design as set covering problem. *IEEE Transactions on Intelligent Transportation Systems*, 17(3):670–679.

[30] Patagón Supercomputer (2021). `https://patagon.uach.cl`. Accessed January 21, 2023.

[31] Peleg, D., Schechtman, G., and Wool, A. (1997). Randomized approximation of bounded multicovering problems. *Algorithmica*, 18(1):44–66.

[32] Peuzin-Jubert, M., Polette, A., Nozais, D., Mari, J.-L., and Pernot, J.-P. (2021). Survey on the view planning problem for reverse engineering and automated control applications. *Computer-Aided Design*, 141:103094.

[33] Ribeiro, C. C., Minoux, M., and Penna, M. C. (1989). An optimal column-generation-with-ranking algorithm for very large scale set partitioning problems in traffic assignment. *European Journal of Operational Research*, 41(2):232–239.

[34] Spensieri, D., Åblad, E., Bohlin, R., Carlson, J. S., and Söderberg, R. (2021). Modeling and optimization of implementation aspects in industrial robot coordination. *Robotics and Computer-Integrated Manufacturing*, 69:102097.

[35] Wang, X. and Ju, S. (2009). A set-covering-based approach for overlapping resource selection in distributed information retrieval. In *2009 WRI World Congress on Computer Science and Information Engineering*, volume 4, pages 272–276.