Diseño y optimización de heurísticas para Minimun Set Cover no ponderado*

Jorge Delgado

Universidad Austral de Chile
jorge.delgado01@alumnos.uach.cl
https://www.uach.cl/

Abstract. As one of the most common NP-hard problem in the research field, many heuristics have been proposed to optimize the solution of the Minimum Set Cover Problem in an attempt to reduce the cardinality or the total cost of finding the sets that make up the solution. In this paper we present an approximation algorithm for an unweighted set cover version, applying a minimal-elements approach along a semi-greedy algorithm to reduce the cardinality of the solution set and accelerate the computational time in succinct space using bitwise operators. Thanks to this technique, our algorithm can process large-scale data at high speed. We test our method with randomly generated data and standard test data available for this problem, achieving at least a 10× speed-up over the classic greedy algorithm and improving the quality of the solution.

Keywords: Set Cover· Greedy Algorithm· NP-hard problem· Approximation Algorithm.

1 Introduction

In the field of computational complexity theory, the study of computational problems allows us to evaluate and classify them according to their difficulty to be solved by a computational algorithm in the so called Class Problems[1][2]. Historically, the optimization of resources needed to fulfill a task is one of the most widely studied problems. Associated to a key objective during the planning process, by searching the balance between cost and results of many human processes. Thus, these problems can be mathematically reduced to a combinatorial problem [3], where many possible solutions can coexist and it is necessary to determine if given a particular input there is a solution, or if there is more than one solution, which of them is the optimal one.
Due to its nature, those combinatorial problems have been classified in the groups of the most difficult problems to be solved by an algorithm. When the time required to execute an algorithm grows exponentially as the input size increases the combinatorial problem is classified as NP when it is a decision problem [1]; or NP-hard when it is an optimization problem [3][2]. The latter are known as the most difficult problems to solve and also to verify any solution. Usually, any effort to find the exact solution for a NP-hard problem implies the use of a non-viable method due to its high computational cost required to solved them, compared to the minimal improvement obtained in many cases.
For this reason, over the last decades the researches has focused on the development of new techniques that, given some particular problem with their own constraints or context, could be roughly solved in polynomial time, i.e. in a bounded period of time, but without necessarily being the most optimal solution. Those techniques are called "approximation algorithms" and allows the computation of hard combinatorial problems in a reduced period of time, compensating with a bounded quality of its solution[4].
The Minimum Set Cover Problem **(MSCP)** is an example of those optimization problems, summarized as: given a target collection of elements and a series of subcollections of itself, the task focus on find the minimum number of subcollections whose union "covers" every element from the initial collection[5][6]. The MSCP is mostly used in industrial planning or scheduling tasks, where a finite set of elements, as number of workers, worker's shifts or transport routes should be coordinated to avoid overlapping or to reduce the operational cost. Even more, strategic placement of minimum necessary telephone antennas or fire stations to cover a city or a specific geographic region is crucial to maximize operational time and minimize response time, both problems that can be modeled as an instance of MSCP.

* This research was partially supported via the Vicerrectoría de Investigación-Universidad Austral de Chile and the InnovING:2030 project through grants: Proyectos de Instalación VIDCA 2020. (VFCI)16ENI2-66903.

1.1 Definition

Given a finite set of elements $U = \{u_1, u_2, ..., u_n\}$, and a collection of subsets of U, $T = \{t_1, t_2, ..., t_m\}$, where $t_i \subset U$ and each element of U is in at least one set of T, the **MSCP** is defined as find the minimum number of sets of T, whose union covers every element in U.
This problem have been proved to be an NP-hard problem, and demonstrated that, under the conjecture $P \neq NP$, an approximated solution can only be found within a factor of $O(\log n)$-approximation [7], and it cannot be lower than $(1 - o(1)) \log n$ unless $P = NP$ [8][9]. Furthermore, the study and development of MSCP is also practical to other problems, since any NP-complete problem can be reduced to any NP-hard problem in polynomial time, if there is a solution to one NP-hard problem in polynomial time, there is a solution to all NP problems in polynomial time.

In this work, we propose two heuristics to find an approximate solution within the factor of $O(\log n)$ mentioned before, obtaining a solution as good as the state-of-the-art approximate algorithm and also reducing the computational time required. This is achieved by using an additional data structure based on the elements of the universe and iterating over a given range to find the best candidate subset to be included. Experimentally we show that our proposal manages to improve both the computation time and the quality of the solution compared to the state-of-art Greedy algorithm[6], using a succinct space approach to represents the elements and bitwise operators to solve each iteration.
In order to introduce the proposed algorithm, we present the most studied approaches in the state-of-the-art, used as baseline to compare our results.

2 Preliminaries

The following topics are based on an deeper understanding of computational complexity theory and therefore complexity classes, for this purpose a few concepts will be clarified to a fully understand of this paper and its implications.

NP Defined as the set of decision problems that can be solved by a non-deterministic Turing Machine in polynomial time. It establish one of the most important set of problems to be studied and solved in the research field and also usually applied to our daily life, where the main question about their input have a yes or no solutions.[1][2][6]

NP-Complete A particular subset of NP problems, it includes the most difficult of those decision problems. The term Complete refers to their capability to simulate any other problem in the same complexity class[1][3]. Therefore, any algorithm that can solve a NP-Complete problem can also be used to solve any other problem in NP, by using a polynomial reduction of them into NP-Complete. For this reason, any improvement in this area take us one step closer to prove (or disapprove) the unsolved question in computed science about if $P = NP$.

NP-Hard Corresponding to a subset of problems that are at least as hard as the hardest problems in NP, the NP-Hard class includes decision and optimization problems that maybe not be in NP (and hence be not decidable). One important property about this class, is that must exist a polynomial reduction from a NP-Complete problem to any NP-hard, similar from NP to NP-complete, and therefore any polynomial-time solution for a NP-hard problem can solve in polynomial-time any other computer science problem, proving $P = NP$[1][3].
To this day, this have not been proved, because there is not an algorithm capable to solve a NP-hard problem in polynomial-time, neither to verified a given solution. That being said, it is possible to find approximate solution, that run in polynomial time, an allows us to find a viable solutions, within some bounds around the theoretical solution. This can be obtained by using Approximation algorithms to our particular scenario.

Polynomial reduction A process to solve a problem A by transforming its input I_a to the input I_b of a problem B, using a polynomial number of computational steps, in addition to a polynomial number of calls of a efficient algorithm to solve B [5]. It also proves that A is at least as easy to solve as B[2][10].

Approximation Algorithms A denomination for algorithms that are formulated to be a way more efficient in time, therefore reducing the computational cost, than their classic or bruteforce-versions[2][6]. The solution proved by those algorithm can be i. A value that is close to the exact solution, or ii. A solution that solves the problem, but is not necessarily the best one. Starting from a simple idea, also called heuristic, those algorithms are allowed to take significant decisions in a particular step, but with less precision than the original version. Deeply linked to search/optimization problems, those algorithms can take smart advantage from some constraint or data trend associated to its problem, proving a solution that is guarantee to be at least as close to the real solution as their approximation ratio or factor.

Succinct Space The use of a particular data structure to contain every element to be processed, using the minimum space in memory needed for this task [11]. This can be achieved by using the exact amount of bits, or at least the lowest number of bytes to contain them.

3 Related Work

On the basis that MSCP is an NP-hard problem, it has been widely studied during decades by mathematicians to reduce the theoretical margin of approximation to its minimum expression that can be later obtained by an algorithm. As in [8], it have been proved that is not possible for a polynomial-time algorithm find a solution with an approximation ratio lesser than $(1 - o(1)) \log n$, unless $P = NP$ is proved. Due to this limitations, the exact solution, in others words the minimum number of subset needed to cover the initial universe, has only been found using an exhaustive search to a limited number of subsets, while the most practical uses of this algorithm have to assume a trade-off between precision and the time required to compute its answer, especially for the case of scheduling jobs or cities building location. In this section, the most reliable and latest approaches in the state-of-the-art will be introduced as an starting point to our research.

3.1 Exhaustive Search of Exact Solution

Traditional version As the simplest approach and widely used as the base algorithm for optimization, the exhaustive search tends to rely on a brute-force approach to find one of the best possible solutions, even if there are more than one. To do that, the exact solution can only be found by evaluating each of the $O(m!)$ possible combinations of the sets of T, with m the number of subsets in T, to determine which of those unions cover each element of the universe. Even when the exponential increment of operations required as the number of subsets increase, implies a high computational cost in many cases, it is the only method to find the exact solution to the problem. From this approach, two important features can be highlighted: First, if we search a solution by combining the minimum numbers of subsets possibles and find one combination that fulfill the MSCP objective, there is no need to test every other possibility; second, as it is necessary that the sum of the cardinalities of the combined sets is at least equal to the universe size, then all cases below that number cannot include all the elements and therefore should not be considered as a possibility.
Those ideas can be used in a new version of exhaustive search that fits better for our purpose, and will define the enhanced exhaustive search used in our experiments.

Enhanced Exhaustive Search As mentioned before, since in a unweighted scenario we only search for the minimum cardinality of one exact solution, even if there are more combinations with the same result, there is no need to try every possible combination after finding the one with the least number of sets involved. For this purpose, we can determine the size of the largest set in T, precomputing the minimum number of sets d needed to equal the number of elements in the universe and iterates every possible combination of d sets $\subset T$ until the union of those selected subsets satisfies the MSCP, or extending d by 1 if neither does.
This algorithm still takes $O(m!)$ time in the worst case, so it is still not an applicable option to many real cases, but improve the average case where the exact solution is more plausible to be found with a lesser number of subsets than m.

Acknowledging the practical issues that both algorithms have, many techniques have been developed to find an approximate solution to the minimum set cover problem, less precise but obtainable in polynomial time that finds the best candidate set in each iteration, over all unselected sets.

3.2 Greedy Set Cover

Based on the popularly **Greedy** heuristic used in the field of optimization, the Greedy set cover is an approximation algorithm developed by Cormen et al.[6], which focus on finding an approximated solution to this hard combinatorial problem by using a polynomial-time algorithm. The method described in **Algorithm 1**, consist in including one set T_d from T at a time, which maximize the number of elements between T_d and the uncovered universe up to that iteration, removing each one of its elements from the universe. It iterates until the universe set is empty.
This idea, allows the procedure to take a naive selection, by covering as much elements of the universe as possible in each step. Compared to the exhaustive search, this version is not able to determine if the sets selected results in the best solution possible.

Algorithm 1 Cormen's Greedy Algorithm

Input
U Finite set of elements to cover.
T A collection of subsets of U whose union is U.
Output
I Set of solution sets.
procedure GREEDY
$I \leftarrow \emptyset$
$X \leftarrow U$
while $X \neq \emptyset$ **do**
Select $S \in T \setminus I$ that maximizes $\left|X \cap S\right|$
$I \leftarrow I \cup \{S\}$
$X \leftarrow X \setminus S$
return I

For MSCP, the Greedy algorithm finds a $O(\log n)$-approximation solution in polynomial time [12]. Due to his deterministic behaviour and simple approach, there are many input cases that force the worst solution possible as output, even when the amount of sets in T is low, by just taking in consideration the number of elements to be covered, as shown in Figure 1, where each step produce an accumulative selection error instead of choose any of the 3 subsets $S3$, $S4$ and $S5$ from the exact solution.

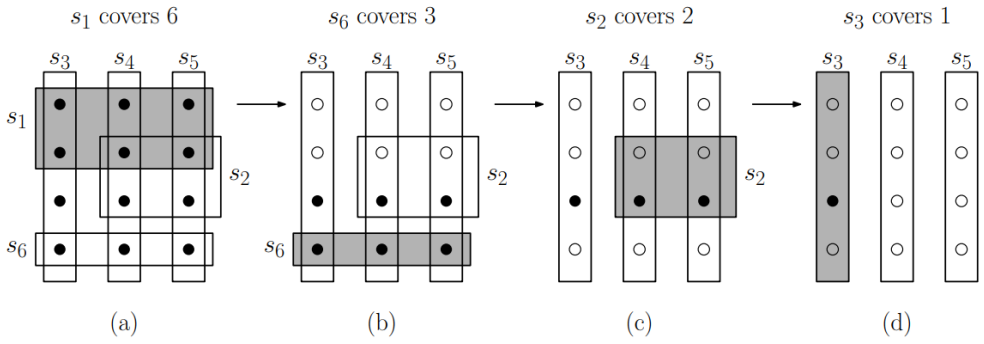

Fig. 1: One possible worst case scenario for Cormen's Greedy Algorithm, in contrast with the 3 columns that represent the exact solution.

As a fast method to find a MSCP approximation, this idea can be improved by reducing the greedy error in each step, implementing a way to prioritize each element or sets over the others, that could correct the naive approach.

3.3 Particular cases

In this section, we will refer to three particular cases derivated from Cormen[6], evaluating their algorithmic approaches over than the efficiency itself due to the different context involved.

Minimum Entropy Set Cover Explored by Eran Halperin and Richard M. Karp[13], they propose an approximation algorithm such as **Greedy Set Cover** to solve under reasonable assumptions a learning problem of identify the distribution and class of a random generated dataset. The use of MSC algorithms allows them to a find highly concentrated cover of the distribution for the data, which in turn has the minimum entropy for the likely class assigned. As they mention, there is no prior work on the complexity theory field to solve this problem in a similar way, working around the Greedy Set Cover with an additional constant cost to solve each instance.
Furthermore, one important assumption is considered for the worst case scenario of Cormen's Algorithm, which is that in a real life problem, a solution most likely includes at least one of the largest set available. Also, this works emphasis in the projection of minimum entropy set cover for computational biology, since it possible to use a cover to compare the match between substring and a original string of binary digits, representing a natural process or DNA sequences.

Eager and Lazy approach In the other side of this problem, Ching Lih Lim et al.[14] points out a few elements that the **Greedy set cover**[6] leaves unspecified in the implementation, such as how to represent the subsets and how to manage each covered elements. For this purpose, two proposal are developed:
i. An Eager algorithm based on **priority queues** supported with **maximum** and **delete** functions to update the sets with the covered elements in each iteration. The Eager behaviour force an continuously update of each subset and uncovered elements of the universe, with a high computational cost in each step of the algorithm and a recurrent call to get the maximum subset from the queue. To avoid those problems and balance the computational cost between recurrent calls and decreasing the weight in each step, their propose an Lazy approach:
ii. If the update of each subsets not included at any point can be delayed until is necessary to check, the algorithm just have to pay, a priori, for the extraction for the subset with the maximum number of elements uncovered. This **Lazy behaviour** still uses the priority queue as the main data structure, but with a new cycle of iteration over the set selected, until it choose the best candidate.

Approximation for K-set Cover Problem As another study case, a k**-SCP** is defined as any instance of the MSCP which each subset of T has at most k elements. This restriction plays an important role as it allows to reduce the approximation ratio of the algorithm to his minimum expression to $(1 + \frac{1}{k})$-approximate of an optimal solution by the use of the **ASC** algorithm developed by Hanaa A. E. Essa et all[15]. Their results are compared to others approximation ratios obtained for k-set covers with basic approaches and in every of them the **ASC** has better results. Nevertheless, the basic representation of the T subsets by a matrix and the use of graphs to represent the neighborhood of subsets has an elevated cost in memory to represent each element of a higher cardinality universe and subsets of it.

Is clearly to see from those proposals that reducing a particular problem to an **MSC** instance, plus the use of some particular heuristic, could achieve a solution quite close to an optimal one. However, that is possible if we focus in the efficiency of solving a subset of MSC instances and not considering the management of memory as an issue; those algorithms are rarely used outside their study case, neither being used as an online processing method or for big data analysis. Also, we have not found others works for the general unweighted MSC problem with a similar approach to our work.

4 Research Question

Given the most actual work on this field, our investigation is focused to answer if it is feasible to implement an algorithm that allows solving the MSC NP-hard problem with an approximation at least as good as the state-of-the-art Greedy algorithm, improving the computation efficiency required to solve it. The established hypothesis is that, by using additional data structures in a preproccesing step, it is possible to achieve an optimized representation of the problem and therefore, obtain an solution as good as Greedy using a lesser amount of computational resources.

5 Our Proposed Algorithm

To treat the challenges exposed before, this work proposes two different techniques to i) attempt to lower the space required by the input sets and ii) reduce the error associated to greedy selection in each iteration. Both tasks are solved by using an element-oriented succinct data structure which was designed directly to support MSCP instances. This allows to represent all elements of U using a total of n bits as a continuous memory space plus a map structure, illustrated in Figure 2, that relates each original integer value to its position in the succinct space $p \in [0..n-1]$. Furthermore,

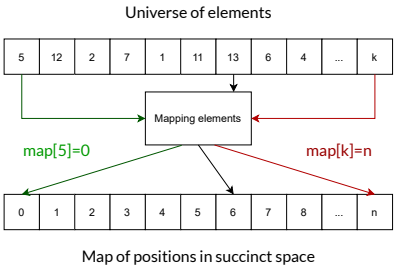

Fig. 2: Representation of mapping elements in succinct space.

each set is represented as an instance of this n-bits space, where each bit $b_i \in T_j$ is set to 1 if $map\,[b_i] \in T_j$, 0 otherwise, as in Figure 3. Our algorithm uses a fast method to compute the intersection between multiples sets by using bitwise operators for each succinct representation and counting the resulting number of ones, improving the processing time obtained. To achieve this representation, we have to allocate $O(n(m+1))$ bits to represent each subset and the universe, plus $O(W * n)$ bits to map each position from the succinct space to each integer representation of the elements, where W is the word size in RAM. Hereafter, the algorithms will work in the described space and using bitwise operations for its purpose.

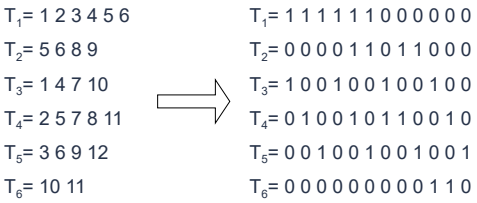

Fig. 3: Transformation of input sets to succinct space.

i. First Heuristic: Minimal Elements In many input scenarios, working with a massive amount of subsets of the universe could turn any greedy approach to select the worse case by searching for the set that maximize the numbers of elements to be covered. For this reason, the first proposed approach focuses on finding the elements u_i in U that are only covered by a unique set $T_j \in T$, as shown in Figure 4; we call these elements as "Minimal elements" for MSCP. Certainly the subset that contains it must be included in any solution, otherwise we will get an incomplete solution.

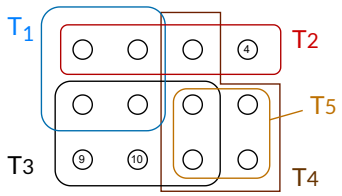

Fig. 4: Minimal elements 4, 9 & 10 can only be covered by one subset of T, respectively.

To be able to recognize and include those minimal elements, we construct an element-oriented data structure (Figure 5), which identifies which sets include each singular element of the universe. This allows the **Algorithm 2** to sort by the number of "repetitions" of each element and, therefore, immediately choose the minimal items at the beginning of this structure.

Value	In set
10	{3}
4	{2}
9	{3}
1	{1, 2}
3	{2, 4}
2	{1, 2}
12	{4, 5}
8	{4, 5}
6	{1, 3}
5	{1, 3}
11	{3, 4, 5}
7	{3, 4, 5}

Fig. 5: Structure used to represent the sorted elements to efficiently identify minimal elements.

Algorithm 2 Minimal Elements

Input
- P Structure of elements to be covered.
- n Size of the universe U.
- S Empty solution set.

Output
- S Subsets that contains each minimal elements in P.

procedure $pre\_set\_cover(P, l, n, S)$
- $sort(P)$
- $pos = 0$
- **while** $P[pos].repetitions == 1$ **do**
  - $S = S \cup P[pos].inSets[0]$
  - $pos = pos + 1$
- **return** S

ii. Second Heuristic: Improved Greedy Approach Given our sorted data structure, which now excludes minimal elements, we consider only the elements not yet included with the least number of repetitions or presences among the available subsets and select a range $s = [l..r]$, where l is the index of the first element not yet covered and r the last element with the same number of repetitions, $0 \leq l \leq r \leq n$. The elements located in that range define our search space for following subset $T_k \in T$ to be included in the solution, which must be the one that covers the largest number of elements not yet included in our partial solution. This process is illustrated in **Figure 6** and described in **Algorithm 3** which is repeated until every element in the range s is covered and then we continue with the next range until there are not uncovered elements in the universe. Note that with this technique we significantly reduce the search space of each iteration, which is restricted only to the subsets defined by the interval s.

Value	In set	
1	{1, 2}	range s with repetitions = 2
3	{2, 4}	
2	{1, 2}	
12	{4, 5}	
8	{4, 5}	
6	{1, 3}	
5	{1, 3}	
11	{3, 4, 5}	range s with repetitions = 3
7	{3, 4, 5}	

Fig. 6: Range of searching with Greedy approach.

Another benefit of this idea is that, in most random cases, the selection of subsets that include less frequent elements will implicitly include many of the most repeated elements. So, we only focus in selecting the set T_k from the range s that covers the highest number of elements in the uncovered universe. By searching over a small selection, instead of all the subsets in T, the searching time is reduced in orders of magnitude compared to Cormen, providing an efficient way to cover each element of the universe during the selection.

Algorithm 3 Improved Greedy Set Cover

Input
- P Sorted structure of elements to be covered.
- l Index of the first element left uncovered.
- n Size of the universe U.
- S Subsets already included in the solution.

Output
- S Subsets whose unions cover every element in U.

procedure $set\_cover(P, l, n, S)$
  while $l \leq n$ **do**
    $rep = P[l].repetitions$
    $set = findCandidate(P, l, rep)$
    $S = S \cup set$
    if $P[l]$ is covered **then**
      $l = l + 1$
  return S

6 Experiments

In this section, we will address the process of comparative testing applied to the baseline algorithms, i.e. Exhaustive Search and Greedy Set Cover, and the proposed techniques presented in this paper, checking both cardinality of solution set and the computation time required. The three algorithms developed in C language use the most appropriated data structure for each case, in order to maximize their performance in a single thread of the CPU.

6.1 Setup

The experiments mentioned were performed in collaboration with the Patagón Supercomputer [16], using a Compute node with 2x AMD EPYC 7742 CPU (2.6Ghz, 64-cores, 256MB L3 Cache), equipped with 1TB RAM DDR4-3200Hz. Even when the machine is GPU dedicated, all experiments were ran only in CPU, with no other significant CPU tasks running and 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. All given runtimes are real (wallclock) times from Chrono library, using the *high_resolution_clock* method.

6.2 Dataset preparation

The data shown below in table 1 correspond to the results obtained by the **Enhanced Exhaustive Search** algorithm, i.e., the exact and optimal solution that gives the minimum cardinality to the problem and therefore it gives us the lower bound for any approximation to the problem. The input data in each experiments involves a series of sets generated randomly by a normal distribution, whose mean was obtained from a uniform distribution over a specified range and with a standard deviation proportional to the expected universe length $n/8$, each test involves a universe's size between $[47..70]$ elements. The second column represent the number of sets generated, the third one correspond to the size of the universe, the forth cardinality of the exhaustive search applied, and the last one the microseconds needed to find the solution. This data allows us to compare both Greedy and our solution in a random scenario, with 5 test for each number of subsets of T.

As a note, the exhaustive search over an amount of 50 subsets was impossible to compute for this experiment, limiting the generated data to what is presented. Since it is not feasible to compute the exact solution for experiments of greater magnitude, subsequent experiments will only consider a comparison between Greedy Set Cover and our algorithm, comparing the cardinality and the time spent in each experiment.

Table 1 Random data generated and processed as baseline.

Exhaustive Search Results				
# Experiment	**# of Sets**	**# Universe**	**Cardinality Solution**	**Processing Time[µs]**
1	20	50	7	251
2		47	7	210
3		48	7	210
4		48	7	210
5		50	7	210
6	25	48	6	499
7		50	6	427
8		50	6	426
9		50	7	529
10		50	7	593
11	30	68	10	83,129
12		68	10	116,871
13		68	13	745,136
14		69	13	704,032
15		70	10	127,763
16	40	70	9	754,720
17		70	10	1,927,608
18		70	10	2,974,182
19		68	12	12,449,053
20		70	10	2,303,615

6.3 Results summary

Experiment 1. With the purpose of compare the performance of the three algorithms discussed before, the Table 2 shows the results obtained from **Greedy Algorithm** and **Our technique** in contrast to the random data generated for its purpose in Table 1. The first column indicates the number of the experiment, followed by the number of sets involved, the size of the universe used and the exhaustive's cardinality obtained, 5th and 6th columns shows the cardinality of the solutions obtained in Greedy and our proposal, respectively. Each experiment was repeated 10 times to estimate properly the computational time required by our machine, as shown in Figure 7.

As in Figure 7 and from now on, all figures related to time complexity of experiments will be shown in logarithmic scale, due to the difference in order of magnitude registered. Furthermore, the exhaustive search shows an asymptotic behavior that diverges with an exponential growth rate from both polynomial algorithms. Those last, maintain a linear growth which shows a visual comparison between the exponential cost carried by the exhaustive search as the number of subsets increase, versus the polynomial time needed for Greedy SC and our algorithm.

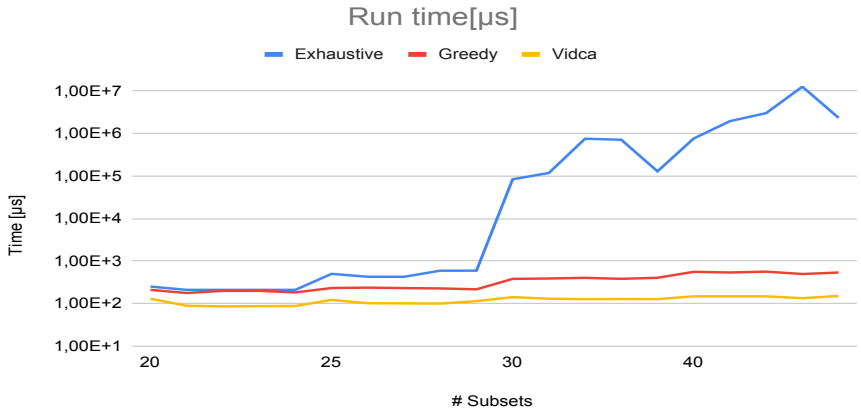

Fig. 7: Comparative time obtained from Exhaustive Search, Greedy SC and our technique.

Table 2 Comparative of experimental results between Exhaustive Search, Greedy SC and our technique.

# Experiment	# of sets	# Universe	Exhaustive	Greedy	Proposed Alg.
1	20	50	7	7	7
2		47	7	7	7
3		48	7	8	7
4		48	7	7	7
5		50	7	7	7
6	25	48	6	8	6
7		50	6	7	7
8		50	6	7	7
9		50	7	7	7
10		50	7	7	7
11	30	68	10	10	10
12		68	10	11	12
13		68	13	14	14
14		69	13	13	13
15		70	10	11	10
16	40	70	9	11	11
17		70	10	11	10
18		70	10	11	10
19		68	12	12	13
20		70	10	11	11

As explained before, since the search for exact solution was not viable for a higher number of subsets, the following experiments only include the comparison between Greedy SC and Vidca Algorithm. The main goal is to prove that VIDCA return a solution at least as good as Greedy for random generated datasets, constructed by the same rules that the data used in experiment 1; and also for natural datasets used in MSC problems.

Experiment 2. Increasing sightly the number of subsets used in each test, the second experiment explore the behavior of **Greedy** and **Vidca** algorithms for **50**, **100** and **200** subsets generated randomly, each one repeated 5 times. The Figure 8 shows the results obtained, with both algorithms staying similar under 60 subsets selected as the better solution, being Vidca at least as good as Greedy in each case. More relevant, the time required by our algorithm is orders of magnitude less than the state-of-the-art, keeping a linear behavior.

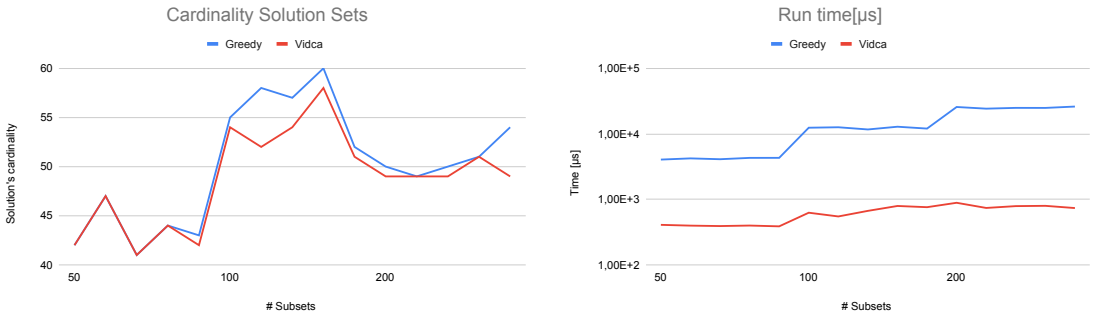

Fig. 8: Comparative results obtained between Greedy SC and our technique for low cardinality sets.

Experiment 3. Using a larger instance of subsets, specifically **300**, **500**, **1.000** and **2.000**, the experiment yields a similar results in cardinality as shown in Figure 9, but with a higher contrast in the time required from Greedy than Vidca. This magnitude of difference tends to increase exponentially due to the greater number of subsets to verify at each step. At 2000 subsets, the speed-up registered by our algorithm was x1000.

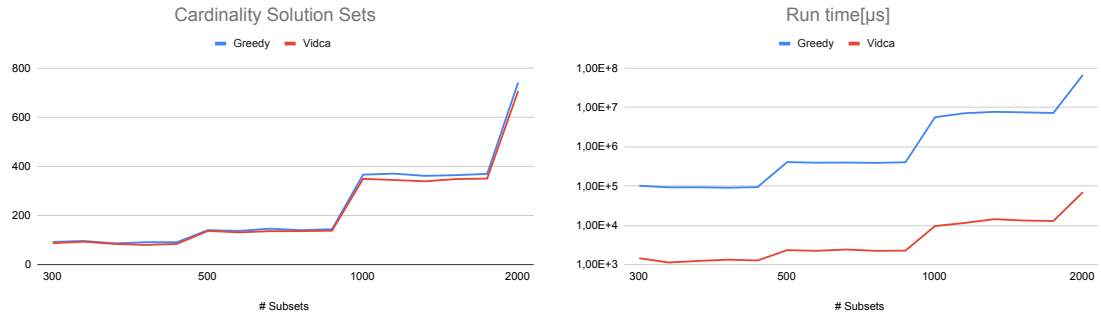

Fig. 9: Comparative results obtained between Greedy SC and our technique for medium cardinality sets.

As a note, experiments of higher magnitude with random data was avoid due to the time required per experiment, and taking those previous results as the basis of improvement achieved by Vidca algorithm.

Experiment 4. As a final test of the implementations, we use a natural dataset (also called real dataset) obtained from the OR-Library [17], which register the combinations of train's routes in order to optimize the number of routes needed to cover each station. Each test has a universe's length between [500..4.500] elements, and between [47.311..102.610] subsets. This dataset is plenty used in Greedy algorithm evaluation, as one of the most normal application of MSC in real life. As results, Vidca keeps recording at least as good as Greedy, with a speed-up between x400 and x16.000.

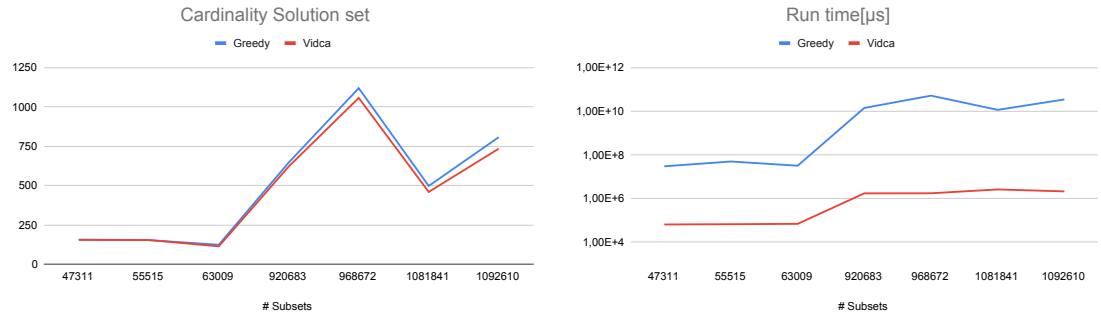

Fig. 10: Comparative results obtained between Greedy SC and our technique for railways dataset.

7 Discussion

Considering the first experiment and the threshold of approximation given by Feige [8], is clearly that none of the approximation algorithms exposed can consistently achieve the exact solution for any MSC instance, especially when the number of subsets involved increase. Nevertheless, since the time required for the exact solution to any instance that involves more than 50 subsets is not feasible, the execution of Greedy or Vidca algorithms in polynomial time allows to obtain at least one viable solution that can be enhanced by repetitions or by the reducing the space of search in each test.

Furthermore, from the results obtained in the remaining experiments, both heuristics developed for our Vidca algorithm has an improvement in the compute of the MSCP compared to Greedy Set Cover, achieving a solution at least as good as the state-of-the-art algorithm and obtaining a speed-up ratio higher than 10 times in many scenarios. This is supported even more by the use of a succinct data structure, which let us processing up to 20 times the amount of data that Greedy would be able to, since the representation of each element is reduced to bitstring and mapped to integers if needed.

Finally, the capability to process unweighted instances of MSC is an important advantage for the state-of-the-art, allowing the processing of multiple real cases with a low computational cost associated in polynomial time.

8 Conclusion

In this paper a new technique is proposed to solve the MSCP which goes hand by hand with the approximation ratio obtained by the most common heuristics in the field of study, and also accelerate the processing speed for each case tested. Furthermore, the use of a bounded amount of memory to represent the data allows its implementation to instances of the problem in larger volume of data, facing the new challenges of cloud computing era as the amount of incoming data keep increasing day by day. As we consider that no other algorithm could solve the problem in a better approximation ratio than has been proved before, it is important to invest in the development of techniques that will cover the basics needs of time and reduction of energy consumption.

8.1 Big data expectations

Given the above statements, it is important to highlight the lately importance of Big Data scenarios that would required the use of more complex and fast algorithms to compute a huge amount of data per experiment. In the cases presented before, the data source is bound to a simple scenario that can afford a large amount of time to be computed, but in the present era of Big Data that would not be rule, quite the opposite. The use of online systems and widely spreed input of data requires an efficient and fast method to respond to these combinatorial problems. In this research, we aim to development of clever data structure for the future implementation of Vidca algorithm for Big Data and Cloud's algorithms, minimizing the memory needed and operations required in each step; and hence using the minimum amount of time for this purpose.

8.2 Worst case scenario

Even when the succinct space technique tends to minimize the memory needed to represent the data, there are certain particular cases that can be compromised comparatively to a trivial decimal representation. Specifically, the worst case scenario for our MSC algorithm would consider a high cardinality of elements in the universe, as well a high cardinality of subsets that only includes a few elements each one. This case force our algorithm to use a large succinct space to represent each one of those subsets, with only a few bits marked as one, which leads to an improper use of the memory and cannot scale as good the experiments presented.

8.3 Future Work

This work seek to establish another heuristic's route for future implementations of unweighted MSC, where an **Compact Data Structure** could support the processing realized in this succinct space, and hence be adopted for more particular MSC cases. As another key stone of our work, not mentioned before since is not in the scope of optimization for MSC, is that the our algorithm is allowed to move back and forth in terms of the elements represented in the succinct space. In other words, we can work with a simple binary representation for each subset all along the process or even reconstruct the original data that each set represent. This is highly important to evaluate each solution or step in a human context, since there can be multiple aspects no considered in the mathematical model. Leaving the decision of deallocating the memory of the map structure used if it is no needed.

In conclusion, the search of improvement in computational complexity theory is a highly demanding topic in our times; by developing new techniques and maximizing the computational resources available we can face a completely new challenges for big data and support the solution for many human problems. Even when the P vs NP problem stills waiting for an answer, every new investigation in this field may takes us one step closer to solve it.

9 Acknowledgments

I'm deeply indebted to Ph.D. Héctor Ferrada, who guided me into this field and help me at every step of this process. Also, special thanks to Ph.D. Cristobal Navarro and Patagon Supercomputer UACH, for allowing to perform our experiments; to Ricardo Aravena & Almendra Mansilla, for their constant support and review of this text; to my parents for everything they have done; and finally, to Fernando Lobos & Felipe Reyes for everything they did.

References

1. Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman; Co., USA, 1990.
2. Ingo Wegener. *Complexity Theory: Exploring the Limits of Efficient Algorithms*. 01 2005.
3. *Np and Np-Completeness*. John Wiley & Sons, Ltd, 1997.
4. Nicolas Bourgeois, Bruno Escoffier, and Vangelis Paschos. Efficient approximation of min set cover by "low-complexity" exponential algorithms. 12 2007.
5. Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.
6. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.
7. Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, September 1994.
8. Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, July 1998.
9. Neal E. Young. *Greedy Set-Cover Algorithms*, pages 379–381. Springer US, Boston, MA, 2008.
10. Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., USA, 2005.
11. Guy Joseph Jacobson. *Succinct Static Data Structures*. PhD thesis, USA, 1988. AAI8918056.
12. V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.
13. Eran Halperin and Richard Karp. The minimum-entropy set cover problem. volume 348, pages 521–544, 07 2004.
14. Ching Lih Lim, Alistair Moffat, and Anthony Wirth. 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, 2014. Australian Computer Society, Inc.
15. Hanaa A. E. Essa, Yasser M. Abd El-Latif, Salwa M. Ali, and Soheir M. Khamis. A New Approximation Algorithm for k-Set Cover Problem. *ARABIAN JOURNAL FOR SCIENCE AND ENGINEERING*, 41(3):935–940, MAR 2016.
16. Austral University of Chile. Patagón supercomputer, 2021.
17. John E Beasley. Or-library, public dataset for optimization problems.
18. Petr Slavík. A tight analysis of the greedy algorithm for set cover. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 435–441, New York, NY, USA, 1996. Association for Computing Machinery.
19. Thao T. Nguyen, Michael R. Souryal, Anirudha Sahoo, and Timothy A. Hall. 3.5 GHz Environmental Sensing Capability Detection Thresholds and Deployment. *IEEE TRANSACTIONS ON COGNITIVE COMMUNICATIONS AND NETWORKING*, 3(3):437–449, SEP 2017.