1.a

Similar to the methodology in [2] and [3] these SCP test cases were generated at random of varying sizes and density. In a later section the results will be compared with that of [2] and [3].

Small example:

```
{0 1 2 3 4 5 6 7 8 9 10}

{
 ({5 8 10 3 2 9 4},1)
 ({9 3 4 2 7 8 5},2)
 ({7 3 1 4 10 0 6 5 8},3)
 ({4 2 1 10 0 9 6 8},1)
 ({3 8 1 5 2 9 0 6 7},2)
 ({0 4 9 8 1 10 2},1)
 ({9 7 8 10 2 3 4},6)
 ({2 8 6 1 10 4},7)
 ({4 0 7 10 5 6 1 3 2},1)
 ({1 6 5 4 3 0},5)
 ({3 8 0 2 7 6 5},6)
}
```

Medium Example:

```
{0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29}

{
 ({27 18 5 1 25 6 20 26 17 2 28 8 4 15},1)
 ({20 13 9 26 0 25 12 14 2 4 24 22},6)
 ({23 7 19 4 16 1 0 25 11 28},2)
 ({5 27 8 16 18 24 10 28 25},9)
 ({27 17 5 2 25 15 13 9 23 6 1 11 18 0 28 8 21 7 10},3)
 ({16 3 13 4 0 26 27 28 9 14 17 18 11 20 15 25 7 8},9)
 ({20 0 13 25 15 8 4 26},6)
 ({10 26 2 22 28 17 21 11 15 16 8 23 29 20 18 13 5 9 4 25},2)
 ({14 3 15 18 20 24 23 10 28 27 6 7 26 0 16 13 9 4 5},5)
 ({1 13 4 15 17 25 5 26 8 7 10},3)
 ({7 21 8 16 26 3 13 27 23 9 4},6)
 ({7 27 19 26 4 2 25 5 18},9)
 ({27 1 28 16 22},6)
 ({16 5 1 9 23 19 4 14 6 25 15 10 17 18 7 8 28 26},3)
 ({8 19 13 1 5 29},10)
 ({8 24 4 28 7 9 13 6 23 17 5 0},10)
 ({13 5 2 3 26 10 12 7 24 8},8)
 ({22 4 7 0 9 13 5 15 26},10)
 ({8 0 3 26 13 5 4 27 7 29 11 28 25},8)
 ({2 14 25 13 11 27 22 28},9)
 ({6 1 29 8 0 27 26 7 5},8)
 ({28 10 26 25 7 4},1)
 ({17 25 7 19 26 12 1},5)
 ({2 13 8 4 10 7 15},10)
 ({28 15 8 3 16 1},4)
 ({22 21 4 19 0 1 27 13 6 9 2 11 26 3 29 28 5 18 7},4)
 ({7 21 1 5 16 0 10 11 20 18 6 22 8 9 25 27 17 28 13},4)
 ({4 12 3 0 11 19 25 20 8 5 10 29 28 18 26},8)
 ({21 24 7 15 12 8 5 19 0 3 27 2 25 4 26 18 13},1)
 ({20 27 7 25 14 18},6)
}
```

There are three other samples that will be tested but are too large to display conveniently here. They will be included in the directory where this file is stored. There is a small, medium and large example of medium density. There is also a medium and large example of sparse density. The following tests are being run on an Intel Core i7 processor @ 2.9GHz with 16GB of RAM. The programming language is Java 14.0.1

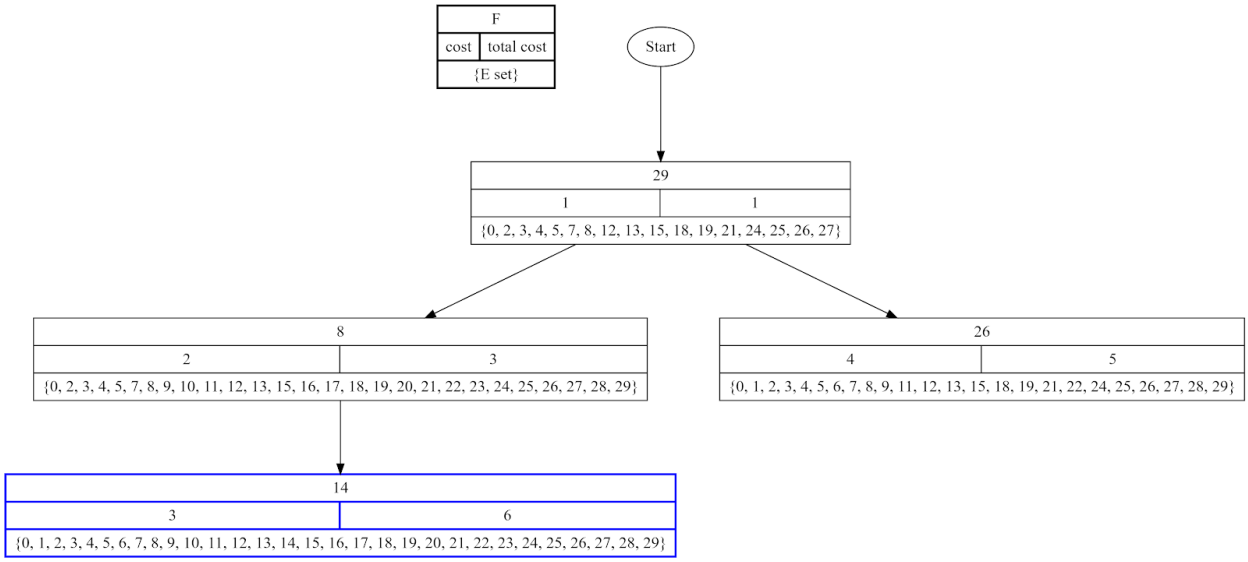
The data from the experiments is as follows:

Size	Number of Columns Before Refinement	Number of Rows Before Refinement	Number of Columns After Refinement	Number of Rows After Refinement	Execution Time	Density
Small	82	11	29	10	168922	Medium
Medium	352	30	202	30	175357	Medium
Large	1820	60	932	60	257654	High
Medium Sparse	98	30	91	30	188281	Sparse
Large Sparse	444	60	434	60	4131421	Sparse

A chart comparing the execution times will be displayed in section 1.c since that is related to time complexity. From this information it can be seen that graphs with a higher density benefit much more from the reduction heuristics than those that are sparse. This intuitively makes sense because a very dense graph will have many overlapping sets in L so the reduction heuristics will

do an effective job at filtering out redundancies. In the sparse graphs however, very little was done by the reductions. The size of the table shrunk by around 1% with the sparse graphs and as much as almost 50% with the most dense. It can also be noted that the execution times for the sparse graphs of identical size to their denser counterparts took longer to complete since the reduction heuristics were not as effective. Not every heuristic will work well in all cases and sparse graphs is clearly a case where these are not very effective. See section 1.c for the chart of execution times.

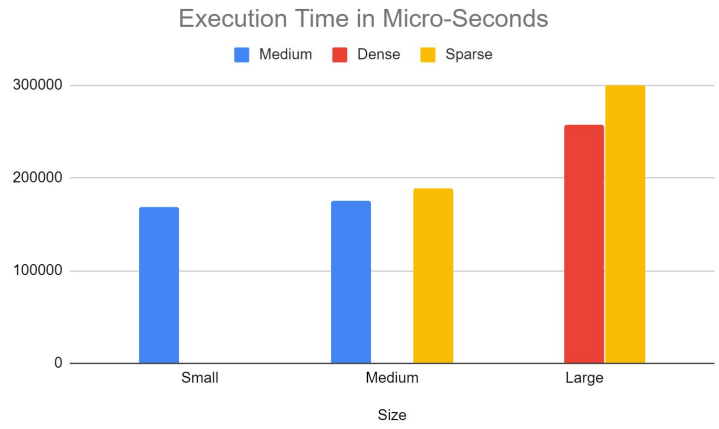
1.b The medium example above produces the following search graph:



There are a small number of search nodes due to the fact that the sets S_j in the set L are large since the graph has a high connectivity. The final optimal solution only uses only three elements of L to cover the set.

1.c The problem domain has a complexity of 2^n. Therefore the implementation will have to be some polynomial multiplied by 2^n. We will examine each of the steps of the algorithm in

Chrisotifes one by one and consider the computational complexity that it will add. Note this is the primitive version without the heuristic refinements.
Step 1: Initializing the tableau should be accomplished in linear time, n, depending on the size of the graph
Step 2: Find P = min[i | $r_i \notin E$] this can also be accomplished in linear time since it only requires iterating through E to find the smallest i that is missing (in reference to r_i)
Step 3: This is the step where we attempt to find the best next set to add to the solution and will be the most computationally expensive 2^n.
Step 4: Backtrack if a dead end is reached or a potential solution is found. In the worst case this will have to climb all the way back up the search graph which should take at most $logn$ time.
Step 5: Is simply setting and checking some variables. This should take constant time.
This will yield a computational complexity of $O(nlogn * 2^n)$
The Only code which compiles and runs is the Java implementation and this code has heuristic refinements already built in therefore the complexity will be different than this theoretical one. However, the section of the code which runs the core SCP algorithm will have the same complexity as stated above. The execution times of the problems from sections 1.a are shown here:



2.a In regard to heuristics for the SCP first we will discuss some a priori reductions which can eliminate redundancies and reduce the sizes of the sets involved. Then once those reductions have been applied we will discuss a heuristic for sorting the rows of the table to further speed up the algorithm. Reduction 1 is stated in Christofides as $if\ \exists r_i \in R\ and\ r_i \notin S_j\ \forall j = 1, ..., N$ then there is no solution and the search should be abandoned. In english this is stating that if there is any element of R, the set we are trying to cover, which is not present in any of the sets in L then it is impossible to cover R with sets from L. Reduction 2 is stated in Christofides as $if\ \exists r_i \in R\ such\ that\ r_i \in S_k\ and\ r_i \notin S_j,\ \forall j \neq k$ then S_k must be in all solutions and we can say that $R = R - \{r_i\}$ and $L = L - \{S_k\}$. This is saying that if there is only one set in L that

contains r_i there is no choice but to use that set, which means that the set and r_i can be removed from consideration. The next two reductions relate to dominance testing and that is not the heuristic we will be covering. After the aforementioned reductions have been applied the next heuristic will be to sort the sets of L in the table to try and create blocks with the least number of columns first. Since the blocks are considered in an ascending order by creating blocks with less columns in the beginning it limits the possible choices and thereby trims the search tree early on.

2.b

Algorithm SCP Approximation

Input: A set R of elements to be covered, A set of sets L and their associated costs
Output: A set of sets from L which covers R with the lowest cost

```
E={}, B={}      //E is the elements covered so far and B is the sets from L being included
Loop
   Find S_i in L such that cost(S_i)/|S_i−E| is minimized      //choose a set from L with best "bang for your buck"
   Add elements of S_i to E
   Add S_i to B
   If E==R break from loop
End Loop
Return B
```

This approximation algorithm is proven in [4] to have a complexity of $logn$. The downside is that although this will always find a solution it will not always find the optimum solution. Especially in large cases where there are many possible solutions this method will likely not converge on the optimum. If all you need for your application is a solution which is good enough then this would be sufficient. But it will not be as thorough as the Christofides algorithm which does exhaust the search space.

2.c Comparing my results from the table in section 1.a to the results in table 3 in [2] the paper from Beasley it is clear that my execution time is much faster. This is without a doubt due to the difference in processing power between 1987 and today. The experiments in [2] were run on a Cray-1S which at the time was very efficient for running this sort of program since it could compute vector addition in constant time. It is referred to as a "supercomputer" in the paper. With a very similar number of rows and columns though, the tests done here show speeds many times faster. Due to the vast difference in hardware it is difficult to make any conclusions about the actual computational complexity and efficacy of his algorithm compared to the Christofides. When comparing the results with [3], a paper from Balas and Ho from 1980, I came to the same conclusion.

3.a I can only speak to the Java implementation of the code since I could not get the C++ or the python to run correctly. The code is severely lacking in comments which makes understanding the programers intention and flow of the program painful and difficult. On a positive note the variable names are clear and well identify what it is meant for as opposed to code which just uses x,y,z.. This helps with ease of understanding despite the lack of comments. The code does not seem very extensible. Meaning that the current implementation is meant to do exactly the one thing it was programmed for and could not be easily added to. The code should have been made in a more modular way so that pieces could be added and removed more easily.

4.a In [5] the authors use an evolutionary algorithm for the set covering problem to find the optimal crew scheduling for commercial airlines. This could easily be extended to the Air Force in many career fields. Assume, for example, that a security forces squadron has K members and that there are sets of these K members which are able to work certain shifts and others who cannot. All members who can work a particular shift are put into a set S_i in L. The set R is the set of all locations on base that need to be manned at a particular time. Given the security forces limited man power and the need to be at all locations simultaneously, find the sets from L that will cover all locations at the least burden to the airmen (cost). This was a real scenario that a cadre member from my ROTC det had to deal with. He found a solution by hand in an attempt to stop his airmen from working 12 hour shifts. Had he been familiar with the SCP perhaps he could have an even better solution in less time!

References

[1]https://en.wikipedia.org/wiki/Set_cover_problem

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

[3]Balas, E., & Ho, A. (1980). Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. In *Combinatorial Optimization* (pp. 37-60). Springer, Berlin, Heidelberg.

[4]http://math.mit.edu/~goemans/18434S06/setcover-tamara.pdf

[5]Marchiori, E., & Steenbeek, A. (2000, April). An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. In *Workshops on Real-World Applications of Evolutionary Computation* (pp. 370-384). Springer, Berlin, Heidelberg.