CS301- Algorithms Project Report

SUBSET SUM

Berk Öztürk, Çağhan Köksal, Furkan Çelik, Hakan Buğra Erentuğ, Nidanur Günay

December 20, 2019

1 Problem Description

Subset Sum is an arithmetic type NP-complete problem. In the subset-sum problem, we are given a finite set S of positive integers and an integer target $t > 0$. We ask whether there exists a subset $S' \subseteq S$ whose elements sum to t. One can encounter the optimization problem associated with this decidability problem in daily life. For example, we may have a truck that can carry no more than t pounds, and n different boxes to ship, the i'th of which weighs x_i pounds. We wish to fill the truck with as heavy as possible without exceeding the given weight limit. [1]

Theorem: Subset sum is NP-complete.

To show that Subset Sum is NP-Complete, we need to show:

1. **Subset Sum problem is in NP:** Given a set S, we need to verify if $\sum_{i \in S} w_i$ is equal to W. Adding w_i's takes up to n many additions and addition can be performed in polynomial time. The solution can be checked for being a solution in $\sum_i O(log w_i)$ time, hence it is in NP.
2. **A known NP-Complete problem can be transformed into Subset Sum problem in polynomial time**: We will show that Subset-Sum problem is as hard as 3-SAT problem.

Proving Subset Sum is in NP-Hard

• Consider a 3-SAT formula with n variables $x_1 x_2, ..., x_n$ and m conjuncts $c_1, ..., c_m$.
• We need to define numbers w_i and a target sum W that is equivalent to this 3-SAT problem.
• We will start by having two numbers a_i and b_i associated with each variable x_i where including a_i will correspond to setting x_i true, and including b_i will correspond to setting x_i false. To do this, let

$$a_i = 10^{m+i} + \sum_{j \ st. \ c_j has \ x_i} 10^j \qquad , when \ x_i = 1$$

$$b_i = 10^{m+i} + \sum_{j\ st.\ c_j has\ \overline{x}_i} 10^j \qquad , when\ x_i = 0$$

• Now, consider an assignment that satisfies 3-SAT and its corresponding a_i's and b_i's:

1. The resulting sum has 1's in the leading n digits, which corresponds to 10^{m+i} for $i = 1, 2, ...n$ since we must include either a_i or b_i for each x_i
2. The resulting sum has either 1 or 2 or 3 (not zero) in the next m digits, since the digit of 10^j represents the exact number of true literals in c_j and we know that assignment is true because it satisfies 3-SAT.

To turn this into a Subset Sum problem we need to define few more numbers. Let

$$W = \sum_{i=1}^{n} +3 \sum_{j=1}^{m} 10^j$$

$$c_j = d_j = 10^j \qquad for\ j = 1, 2, ...m$$

• We will claim this subset sum is solvable if and only if the 3-SAT is satisfiable. Assume that the given 3-SAT assignment is satisfiable, then we select the number a_i when $x_i = 1$ and select b_i when $x_i = 0$. So, we find the leading n digits of W correct.
• To calculate the remaining m digits of W, we will need to add c_j or $c_j + d_j$ depending on the correct number of x's (3,2 or 1) in each clause. We can only get the total of W by including the right number of 1s in any digits, otherwise it will cause a carry and we will not be able to find W.
• To do this , one needs to include exactly one of a_i or b_i , and the corresponding truth assignment needs to satisfy the formula. This means to solve this particular subset sum problem, we need to know satisfiability of corresponding 3-SAT formula. We also observe that this transformation can be done in polynomial time. This means Subset Sum is as hard as 3-SAT and thus subset sum is also an NP-Complete problem. [2]

2 Algorithm Description & Algorithm Analysis

2.1 Exponential Time Algorithm as the Baseline

Definition: An algorithm with integer inputs $a_1, a_2, ..., a_k$ is a polynomial-time algorithm if it runs in time polynomial in $lga_1, lga_2, ..., lga_k$, that is, polynomial in the lengths of its binary encoded inputs. [1]

Assume we find each subset S' of S and then find the sum of elements in each S' . Then among the subsets whose sum does not exceed t, we choose the one with the closest sum. To do this, the algorithm will take an input set $S = x_1, x_2, x_3, ...x_n$ and a target value t. Then, it iteratively computes L_i, the list of all subsets of $x_1, ...x_i$ that do not exceed t, and then it returns the maximum value in L_n. During this process, the length of L_i can grow as large as 2^i, so *exactSubsetSum* is an exponential-time algorithm. (Unless in the special cases in which t is polynomial in $|S|$ or all the numbers in S are bounded by a polynomial in $|S|$.) [1]

For the implementation of exactSubsetSum, we need the following auxiliary functions :

```
# returns the list of integers derived from L by increasing each element of L by
    number.
def plus(l,number):
  li=l
  n=len(li)
  for i in range(0,n):
    li[i]+=number
  return li

#returns the sorted list that is the merge of its two sorted input lists L and L0
    with duplicate values removed.
def mergeLists(li1,li2):
  x1=0
  x2=0
  ret=[]
  n1=len(li1)
  n2=len(li2)
  while(x1!=n1+1 and x2!=n2+1):
    if(x1==n1):
      ret = [*ret, *li2[x2:n2]]
      break
    if(x2==n2):
      ret = [*ret, *li1[x1:n1]]
      break

    if(li1[x1]<=li2[x2]):
      ret.append(li1[x1])
      x1+=1
    else:
      ret.append(li2[x2])
      x2+=1
  return ret

#remove each element in list li that is larger than t
def removeEach(li,t):
    n=len(li)
    ret=[]
    for i in li:
        if(i<=t):
            ret.append(i)
    li=ret
    return li

def exactSubsetSum(li,t):
  n=len(li)
  L=[[]]*(n+1)
  L[0]=[0]
  for i in range(1,n+1):
    L[i]=mergeLists(L[i-1].copy(),plus(L[i-1].copy(),li[i-1]))
    removed =removeEach(L[i],t)
    L[i] = removed
  return max(L[n])
```

2.2 A polynomial-time approximations Algorithm

Main Idea: If two values in L are close to each other, then since we want just an approximate solution, we do not need to maintain both of them in L.
We can "trim" each L[i] list, after creation. (after line 47 in exact_subset_sum)
Trimming a list L by δ means,
Define δ (**the trimming rate**) such that $0 < \delta < 1$
Consider the following inequality for elements y and z in list L:

$$\frac{y}{1+\delta} \leq z \leq y$$

If ,for element y, there exist an element z that satisfies the inequality above, it means that z approximates y within the range provided by δ. Therefore, we remove y from list L.
Else, y remains in the list.

So, we will be able to decrease the number of elements kept in list while having representatives in the list for removed elements.

The trimming function is below. Given that L is monotonically increasing, *trim* trims the list in $\Theta(m)$ time and outputs a trimmed and sorted list. With the help of trim function, we build approximateSubsetSum method on top of exactSubsetSum by following these steps: As new input to method, let the user to defineϵdetermine the allowed range of the approximate subset-sum solution. Hence, approximate solution is allowed to be within a $1+\epsilon$ factor of optimal solution. To realize this, set δ to $\frac{\epsilon}{2n}$ and *trim* L_i at each iteration of the main loop.

The implementation of trim and approximateSubsetSum is shown below:

```
def trim(L,delta): # delta: the trim rate
  n=len(L)
  L2=[L[0]]
  last=L[0]
  for i in range(1,n):
     if(L[i]>last*(1+delta)):
        L2.append(L[i])
        last=L[i]
  return L2

def appSubSet(S,t,e):
    n=len(S)
    L=[[]]*(n+1)
    L[0]=[0]
    for i in range(1,n+1):
          L[i]=mergeLists(L[i-1].copy(),plus(L[i-1].copy(),S[i-1]))
          L[i]=trim(L[i],e/(2*n))
          removed = removeEach(L[i],t)
          L[i] = removed
    return max(L[n])
```

Theorem:
Approximation subset sum is a fully polynomial-time approximation scheme for the subset-sum problem.

Proof:
Trimming function removes every element that is greater than t and that operation maintain the trait that each elements in L_i is also an element in P_i.
Let call the return value of the function *appSubSet* that is the maximum element in the merged list as z^* as. Let $y^* \in P_i$ and it is the optimal solution for the subset sum problem. Since we remove the elements that is greater than t, $z^* \leq y^*$.

We must show that
$$\frac{y^*}{z^*} \leq 1 + \epsilon$$
and running time of this algorithm is polynomial in both $\frac{1}{\epsilon}$ and the size of the input.

Every element in y in P_i is at most t and there exist an element $z \in L_i$ such that
$$\frac{y}{(1 + \epsilon/2n)^i} \leq z \leq y$$

This inequality must also hold for $y^* \in P_n$, and there exist an element $z \in L_n$ such that
$$\frac{y^*}{(1 + \epsilon/2n)^i} \leq z \leq y^* \text{ and therefore } \frac{y^*}{z} \leq (1 + \frac{\epsilon}{2n})^n$$
.

Since we know that $z \in L_n$ fulfilling the inequality $\frac{y^*}{z} \leq (1 + \frac{\epsilon}{2n})^n$, inequality must holds for z^* that is the largest value in L_n.

Thus, our inequality becomes as $\frac{y^*}{z^*} \leq (1 + \frac{\epsilon}{2n})^n$.

In that step of proof, we need to show that $\frac{y^*}{z^*} \leq 1 + \epsilon$. In order to prove that inequality, we supposed to show that $(1 + \frac{\epsilon}{2n})^n \leq 1 + \epsilon$. Since $\lim_{x \to \infty} (1 + \frac{\epsilon}{2n})^n = e^{\epsilon/2}$ and $\frac{d}{dn}(1 + \frac{\epsilon}{2n})^n < 0$, $(1 + \epsilon/2n)^n$ increases with n as it approaches it's limit.

We should note that $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... = \sum_{i=0}^{\infty} \frac{x^i}{i!}$ (Taylor series).Thus $e^x \geq 1 + x$, and that inequality holds only when $x = 0$. When $|x| \leq 1$ we have the approximation $1 + x \leq e^x \leq 1 + x + x^2$.

$$(1+\frac{\epsilon}{2n})^n \leq e^{\epsilon/2}$$
$$(1+\frac{\epsilon}{2n})^n \leq 1+\epsilon/2+(\epsilon/2)^2$$
$$(1+\frac{\epsilon}{2n})^n \leq 1+\epsilon$$

Combination of $\frac{y^*}{z^*} \leq (1+\frac{\epsilon}{2n})^n$ and $(1+\frac{\epsilon}{2n})^n \leq 1+\epsilon$ we've achieved to prove that $\frac{y^*}{z^*} \leq 1+\epsilon$

In order to show that Approximation Subset Problem is a fully polynomial time approximation scheme, we derive a bound on the length of L_i. Let's call the successive elements of L_i as z and z' after trimming and we know that $\frac{z'}{z} > 1+\frac{\epsilon}{2n}$.Thus, They can differ by factor of at least 1+ $\epsilon/2n$.
Therefore, each list contains the value 0, possibly the value 1, and up to $\lceil \log_{1+\epsilon\ 2n} t \rceil$ additional values. The number of elements in each list L_i is at most

$$\log_{1+\epsilon/2n} t + 2 = \frac{\ln t}{\ln(1+\epsilon/2n)} + 2$$

We should remember that Taylor series approximation of $\ln(1+x)$ is $x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + ...$
When $x > -1$, we get the inequalities that

$$\frac{x}{1+x} \leq \ln(1+x) \leq x$$

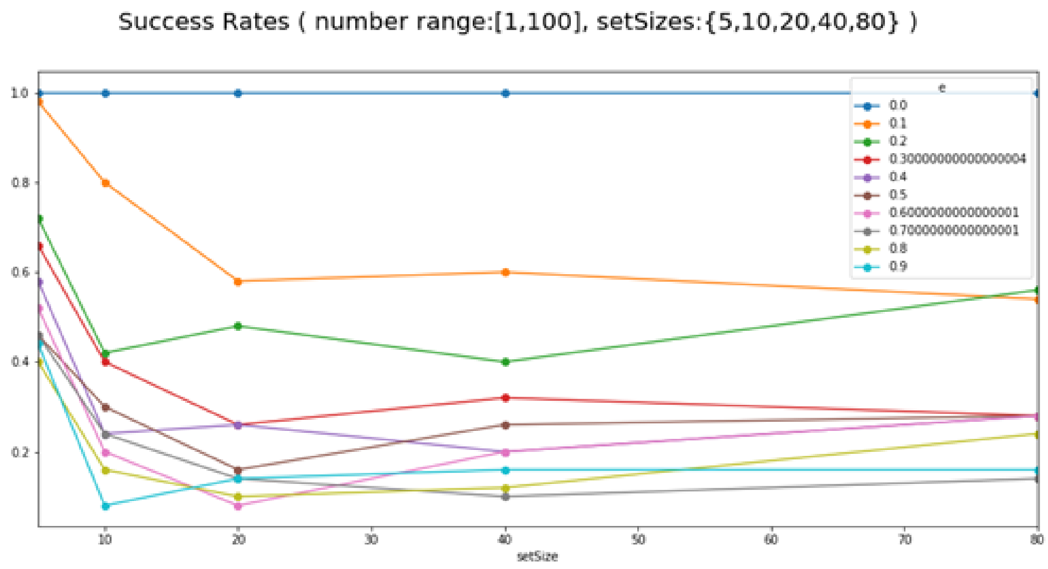
Thus,

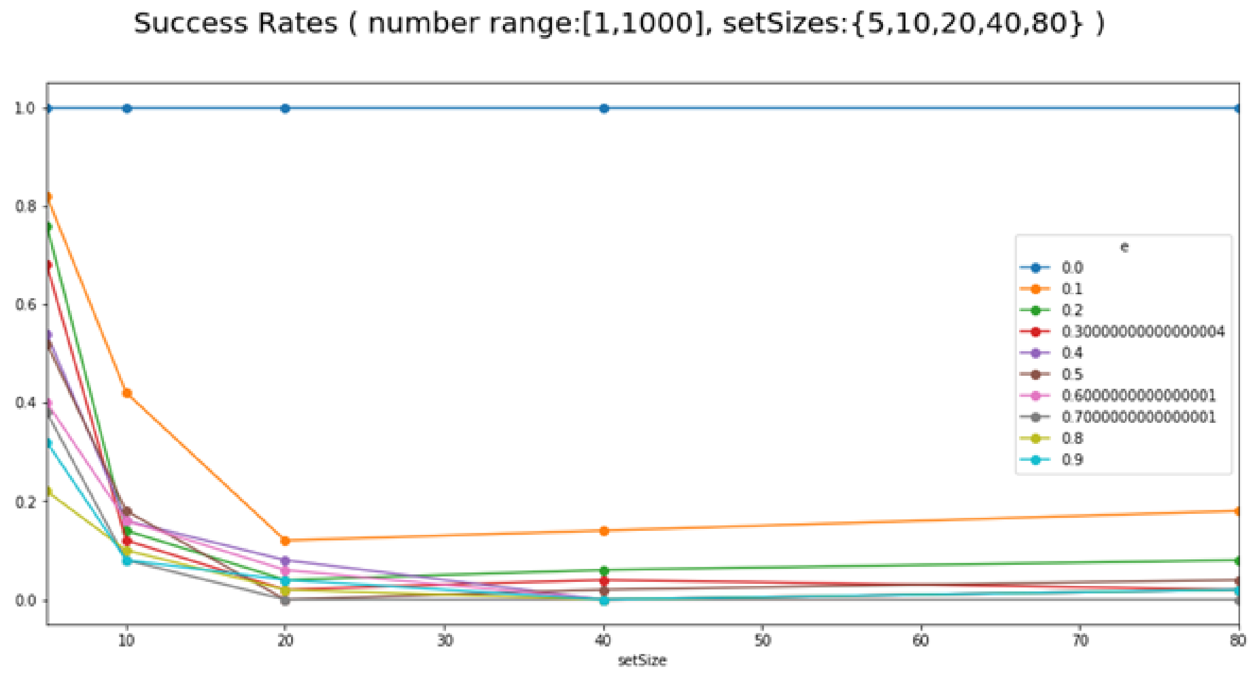
$$\frac{\ln t}{\ln(1+\epsilon/2n)} + 2 \leq \frac{2n(1+\epsilon/2n)\ln t}{\epsilon} + 2 < \frac{3n \ln t}{\epsilon} + 2$$

The number of bits in input lg$t$ supposed to be represent as t plus the number of bits needed to represent the set S and therefore, this bound is polynomial in the size of the input and in 1ϵ.
We've proved that running time of Approximation Subset Sum is polynomial in the length of the L_i. As a result, Approximation Subset Sum is a fully polynomial time approximation scheme.

3 Experimental Analysis

For experimental analysis, we randomly generated some sets (with sample size $n =$ 50) in sizes of 5, 10, 20, 40 and 80 in the ranges of [1-100], [1-1000] and [1-10000] to inspect behaviour of algorithm on different input values.





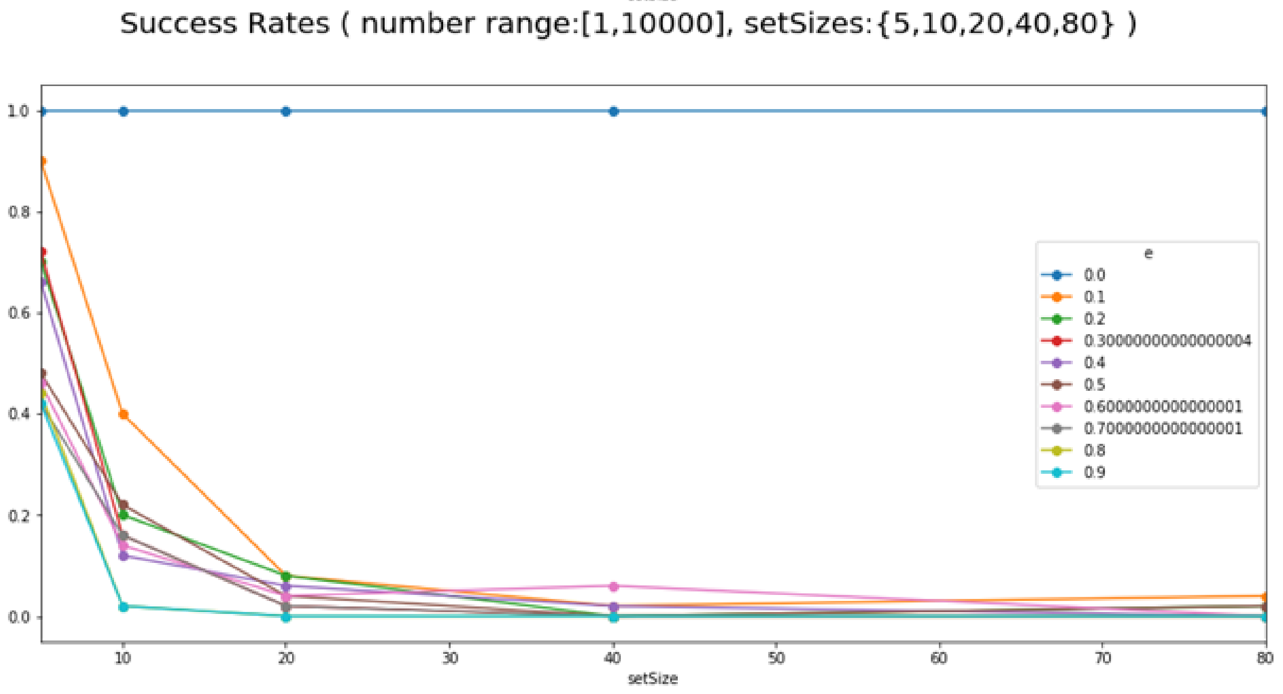


Figure 1: Success Rates for sets with numbers in range [1,100],[1,1000] and [1, 10000]

3.1 Success Rate

When analyzing the success rate of the algorithm, we defined the success criteria as follows:
we used the answer of exactSubsetSum function as the correct answer and accepted that approxSubsetSum is correct if and only if it gives the correct answer. Although the difference between approximate and correct answers is very small, it is considered as failure. Then, we get the ratio of runs giving the correct sum over total number of runs . (For the given ϵ and setSize values)

By looking at the graphs we observed that, as number of elements or the upper bound of the range of set elements increase, success rate decreases for all cases.

3.2 Execution Time

In order to analyze our algorithm's execution time, we established some helper functions with Python libraries Pandas and Scipy and Matplotlib. Then we:

- sampled random number of elements from the set and calculated the sum of the "test-subset" in a helper function.
- measured the execution time and checked if the approximate answer is within the $|1 + \epsilon|$ bound.
- tabulated experiment results with respect to setSize .
- calculated mean and standard deviation of results.
- constructed a 2 sided Confidence Interval with % 95 Confidence Level.

```python
#Find the desired t-score depending on the Confidence Level desired to build

from scipy.stats import t
dof=50-1 # n=50 for our experiments
alpha = 0.025 # means we construct a %95 CI (2-sided )
#alpha = 0.05 # means we construct a %90 CI (2-sided )
t_score=t.ppf(1 - alpha, df=dof)

#Getting CI tables for a given t-score
df_mean= df_range100.groupby(['setSize', 'e'])['execution_time'].mean().unstack()
df_est_std_error= df_range100.groupby(['setSize', 'e'])['execution_time'].std().
    divide(np.sqrt(n_samples)).unstack()
upper_95= df_mean + df_est_std_error.multiply(t_score)
lower_95= df_mean - df_est_std_error.multiply(t_score)
CI_95_range100=lower_95.round(5).astype(str).add('-').add(upper_95.round(5).astype(
    str))
CI_95_range100
```

e	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
setSize										
5	0.0001-0.00012	9e-05-0.0001	0.0001-0.00011	9e-05-0.0001	9e-05-9e-05	8e-05-9e-05	9e-05-0.0001	8e-05-9e-05	8e-05-9e-05	8e-05-9e-05
10	0.00105-0.00136	0.00083-0.00108	0.0008-0.00094	0.00068-0.00077	0.00055-0.00067	0.00052-0.0006	0.0004-0.00049	0.00042-0.00049	0.00041-0.00046	0.00038-0.00045
20	0.0104-0.01357	0.00981-0.01181	0.00678-0.00806	0.00499-0.00599	0.00397-0.0047	0.00379-0.00433	0.00325-0.00372	0.00275-0.00325	0.00265-0.00304	0.00248-0.00286
40	0.0496-0.06666	0.04335-0.0553	0.03679-0.04366	0.02852-0.03428	0.03324-0.04788	0.02054-0.02443	0.01854-0.02124	0.0176-0.01993	0.01666-0.01868	0.0148-0.01681
80	0.20899-0.27605	0.20529-0.2565	0.14932-0.18435	0.12763-0.15245	0.10657-0.13047	0.08683-0.10515	0.07901-0.09705	0.08305-0.09369	0.07403-0.08442	0.06809-0.0787

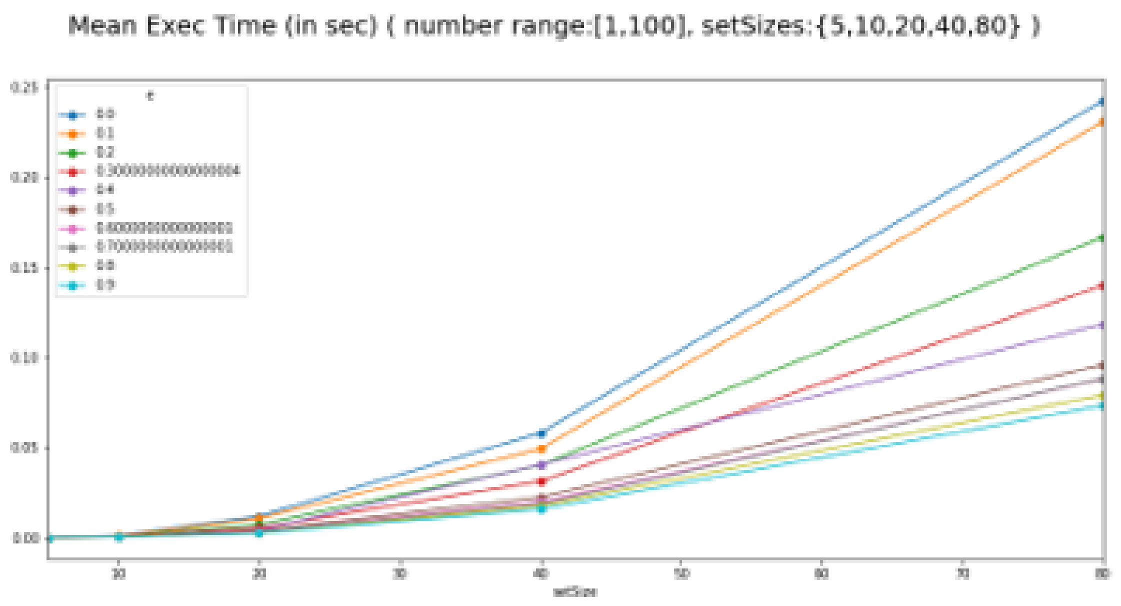
Figure 2: % 95 Confidence Interval for sets with numbers in range [1,100]

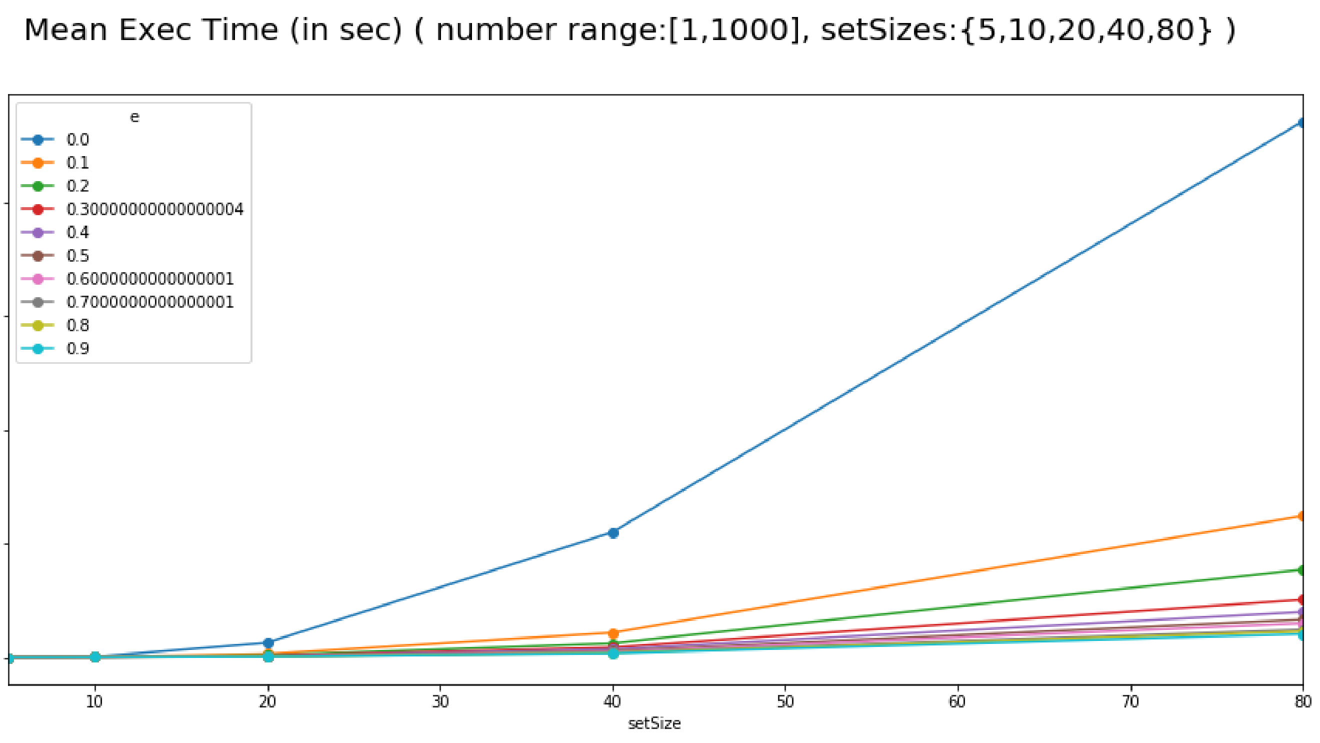
e	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
setSize										
5	9e-05-0.0001	8e-05-0.0001	8e-05-9e-05	9e-05-9e-05	8e-05-9e-05	8e-05-9e-05	8e-05-9e-05	8e-05-9e-05	7e-05-8e-05	7e-05-8e-05
10	0.00155-0.00199	0.00088-0.00111	0.00065-0.00081	0.00056-0.00067	0.00047-0.00057	0.00046-0.00054	0.00042-0.00049	0.00042-0.00048	0.0004-0.00047	0.00036-0.00042
20	0.06146-0.08567	0.0117-0.0139	0.00715-0.00856	0.00559-0.00651	0.00434-0.00501	0.00361-0.00436	0.00314-0.00371	0.00291-0.00337	0.00259-0.00298	0.00264-0.0029
40	0.48092-0.62375	0.0999-0.11693	0.06098-0.06926	0.03976-0.04623	0.03218-0.03746	0.02658-0.0299	0.02174-0.02547	0.02076-0.02294	0.01782-0.01995	0.01496-0.0173
80	1.90526-2.5558	0.53742-0.63845	0.35578-0.39163	0.2423-0.27677	0.18412-0.20907	0.15259-0.17682	0.12677-0.14787	0.12374-0.13521	0.10841-0.11696	0.09892-0.10722

Figure 3: % 95 Confidence Interval for sets with numbers in range [1,1000]

e	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
setSize										
5	9e-05-0.00011	9e-05-0.0001	9e-05-9e-05	9e-05-0.0001	8e-05-9e-05	8e-05-9e-05	8e-05-9e-05	8e-05-9e-05	8e-05-8e-05	7e-05-8e-05
10	0.00161-0.00206	0.00083-0.00104	0.00066-0.00081	0.00062-0.00071	0.0005-0.0006	0.00044-0.00053	0.00044-0.00052	0.00042-0.00048	0.00038-0.00044	0.00036-0.00041
20	0.27847-0.41898	0.01177-0.01457	0.00751-0.00894	0.00492-0.00602	0.00402-0.00486	0.0033-0.00407	0.00292-0.00352	0.00283-0.00327	0.00244-0.0029	0.00247-0.00278
40	4.01816-5.29636	0.09874-0.11906	0.05788-0.06762	0.03727-0.04526	0.0319-0.037	0.02528-0.02981	0.02228-0.02603	0.01949-0.02208	0.01783-0.02001	0.01657-0.01855
80	18.6183-24.95255	0.71381-0.81735	0.37477-0.42362	0.24761-0.28462	0.20861-0.22602	0.17317-0.18599	0.12924-0.14824	0.12276-0.13552	0.11034-0.121	0.09314-0.10491

Figure 4: % 95 Confidence Interval for sets with numbers in range [1,10000]





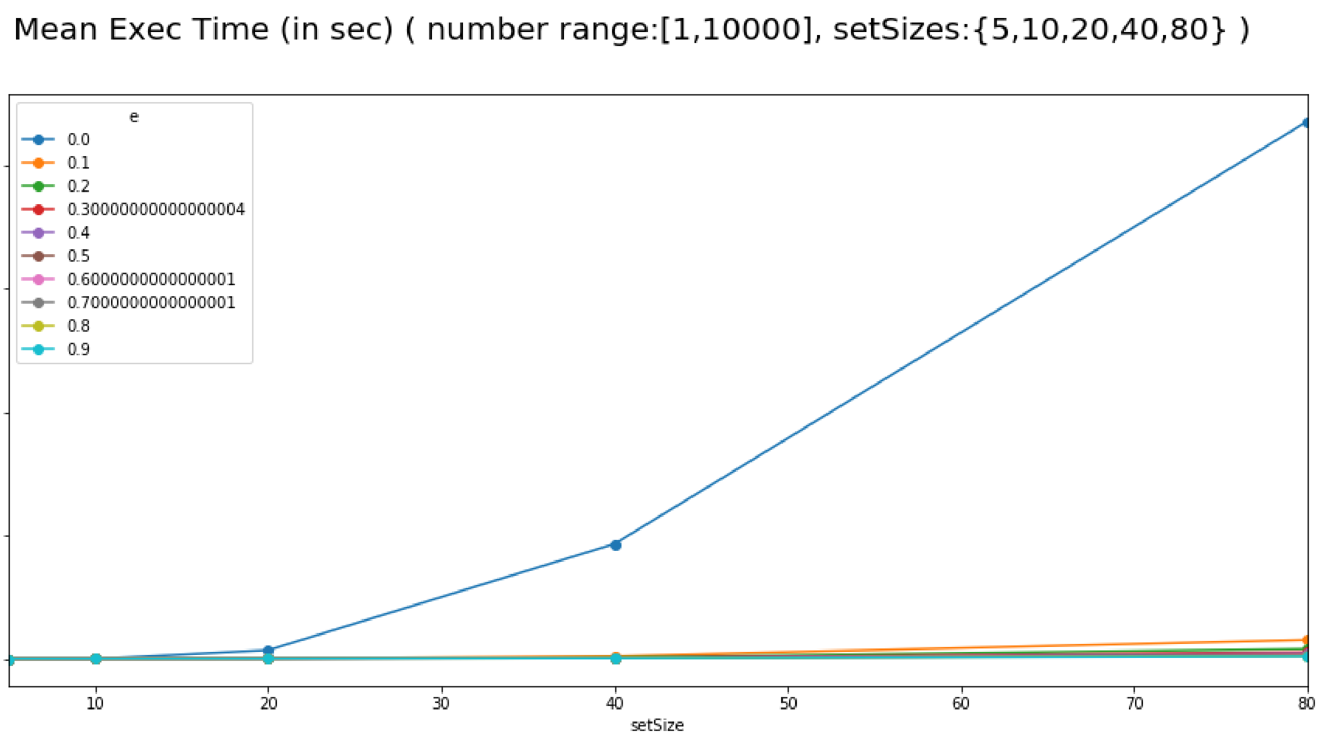
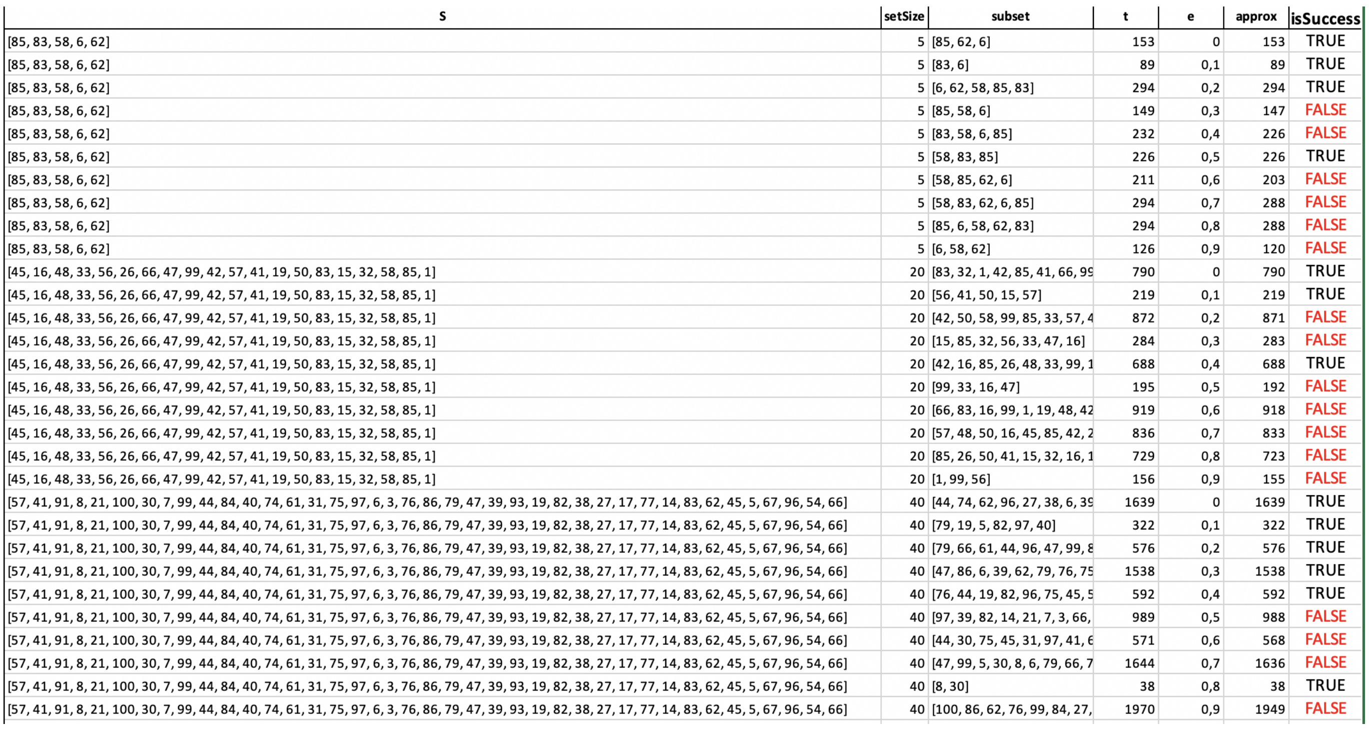


Figure 5: Mean execution time for different set sizes

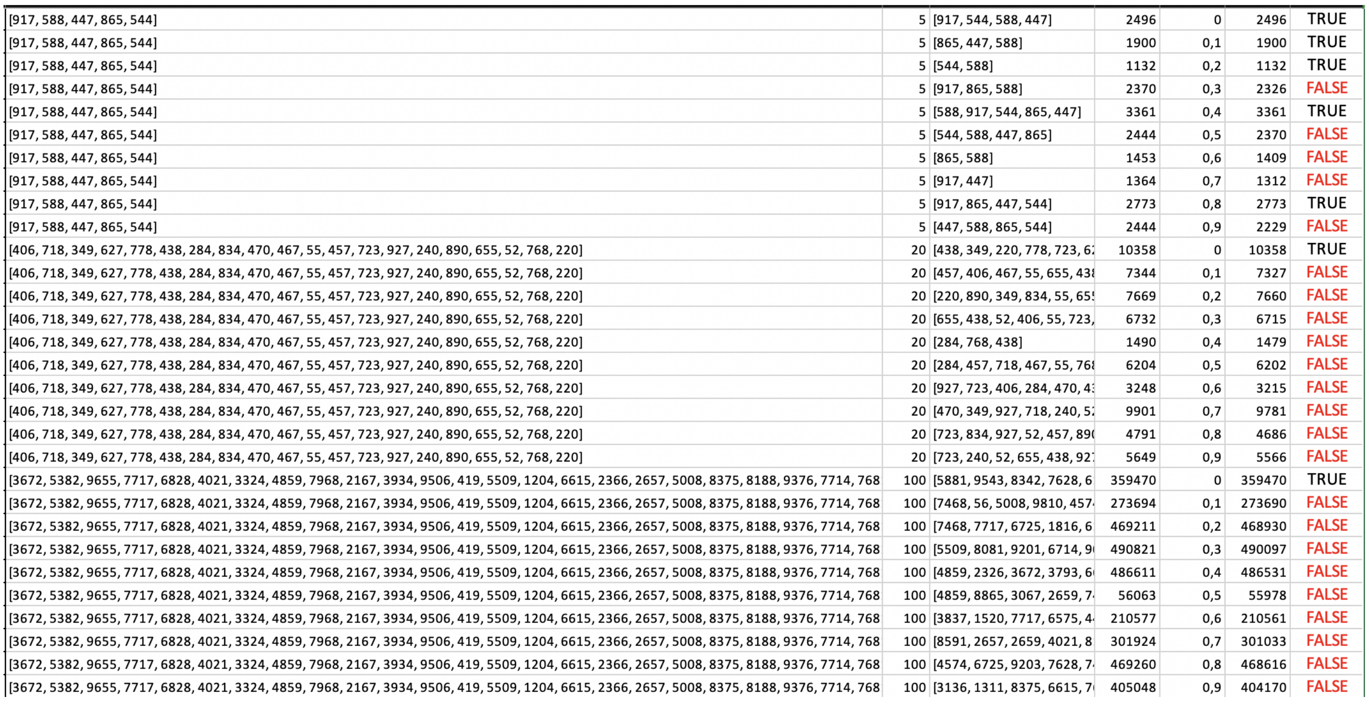
4 Testing

Black-box testing method is used in order to test our algorithm on randomly selected set elements with different set size,different range and different error rates. The test cases and the test results are in figure below:



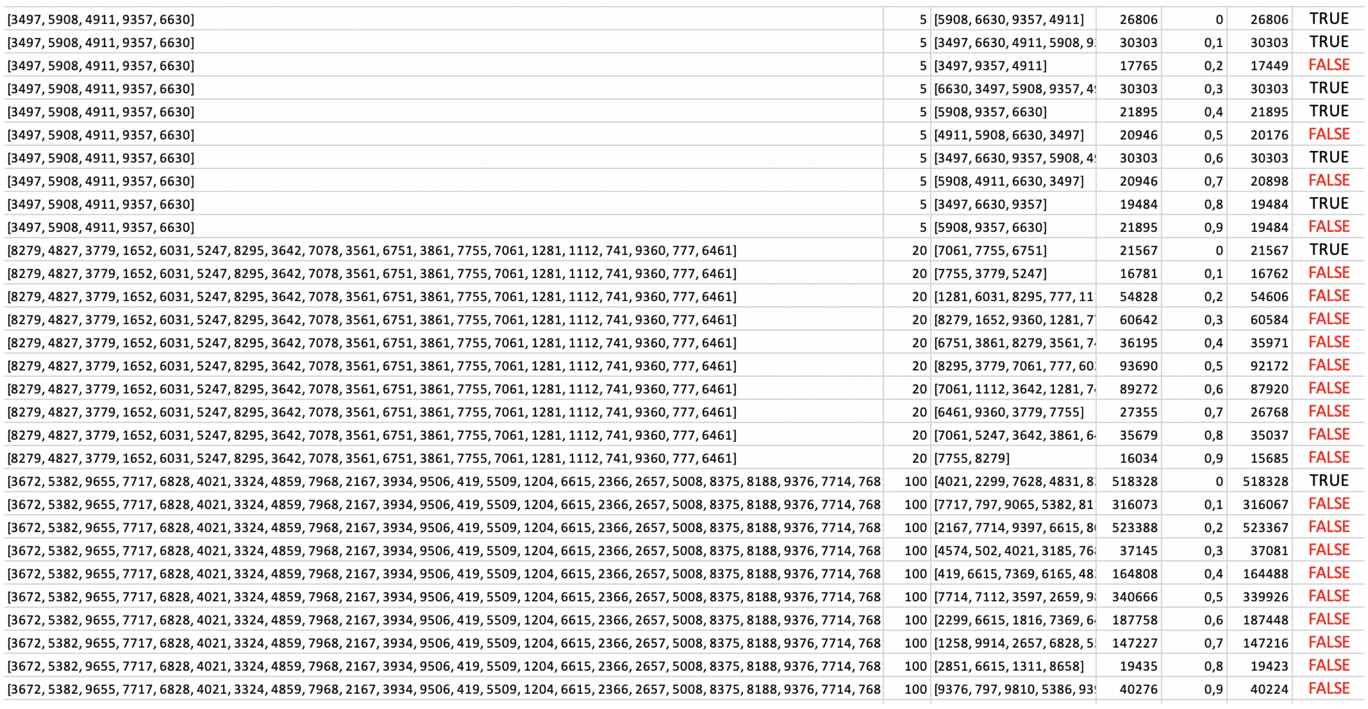
S	setSize	subset	t	e	approx	isSuccess
[85, 83, 58, 6, 62]	5	[85, 62, 6]	153	0	153	TRUE
[85, 83, 58, 6, 62]	5	[83, 6]	89	0,1	89	TRUE
[85, 83, 58, 6, 62]	5	[6, 62, 58, 85, 83]	294	0,2	294	TRUE
[85, 83, 58, 6, 62]	5	[85, 58, 6]	149	0,3	147	FALSE
[85, 83, 58, 6, 62]	5	[83, 58, 6, 85]	232	0,4	226	FALSE
[85, 83, 58, 6, 62]	5	[58, 83, 85]	226	0,5	226	TRUE
[85, 83, 58, 6, 62]	5	[58, 85, 62, 6]	211	0,6	203	FALSE
[85, 83, 58, 6, 62]	5	[58, 83, 62, 6, 85]	294	0,7	288	FALSE
[85, 83, 58, 6, 62]	5	[85, 6, 58, 62, 83]	294	0,8	288	FALSE
[85, 83, 58, 6, 62]	5	[6, 58, 62]	126	0,9	120	FALSE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[83, 32, 1, 42, 85, 41, 66, 99	790	0	790	TRUE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[56, 41, 50, 15, 57]	219	0,1	219	TRUE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[42, 50, 58, 99, 85, 33, 57, 4	872	0,2	871	FALSE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[15, 85, 32, 56, 33, 47, 16]	284	0,3	283	FALSE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[42, 16, 85, 26, 48, 33, 99, 1	688	0,4	688	TRUE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[99, 33, 16, 47]	195	0,5	192	FALSE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[66, 83, 16, 99, 1, 19, 48, 42	919	0,6	918	FALSE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[57, 48, 50, 16, 45, 85, 42, 2	836	0,7	833	FALSE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[85, 26, 50, 41, 15, 32, 16, 1	729	0,8	723	FALSE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[1, 99, 56]	156	0,9	155	FALSE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[44, 74, 62, 96, 27, 38, 6, 39	1639	0	1639	TRUE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[79, 19, 5, 82, 97, 40]	322	0,1	322	TRUE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[79, 66, 61, 44, 96, 47, 99, 8	576	0,2	576	TRUE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[47, 86, 6, 39, 62, 79, 76, 75	1538	0,3	1538	TRUE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[76, 44, 19, 82, 96, 75, 45, 5	592	0,4	592	TRUE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[97, 39, 82, 14, 21, 7, 3, 66,	989	0,5	988	FALSE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[44, 30, 75, 45, 31, 97, 41, 6	571	0,6	568	FALSE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[47, 99, 5, 30, 8, 6, 79, 66, 7	1644	0,7	1636	FALSE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[8, 30]	38	0,8	38	TRUE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[100, 86, 62, 76, 99, 84, 27,	1970	0,9	1949	FALSE

Figure 6: Randomly chosen set elements in range [0-100] and set size of [5,20,40] with different error rates



[917, 588, 447, 865, 544]	5	[917, 544, 588, 447]	2496	0	2496	TRUE
[917, 588, 447, 865, 544]	5	[865, 447, 588]	1900	0,1	1900	TRUE
[917, 588, 447, 865, 544]	5	[544, 588]	1132	0,2	1132	TRUE
[917, 588, 447, 865, 544]	5	[917, 865, 588]	2370	0,3	2326	FALSE
[917, 588, 447, 865, 544]	5	[588, 917, 544, 865, 447]	3361	0,4	3361	TRUE
[917, 588, 447, 865, 544]	5	[544, 588, 447, 865]	2444	0,5	2370	FALSE
[917, 588, 447, 865, 544]	5	[865, 588]	1453	0,6	1409	FALSE
[917, 588, 447, 865, 544]	5	[917, 447]	1364	0,7	1312	FALSE
[917, 588, 447, 865, 544]	5	[917, 865, 447, 544]	2773	0,8	2773	TRUE
[917, 588, 447, 865, 544]	5	[447, 588, 865, 544]	2444	0,9	2229	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[438, 349, 220, 778, 723, 6	10358	0	10358	TRUE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[457, 406, 467, 55, 655, 43	7344	0,1	7327	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[220, 890, 349, 834, 55, 65	7669	0,2	7660	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[655, 438, 52, 406, 55, 723,	6732	0,3	6715	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[284, 768, 438]	1490	0,4	1479	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[284, 457, 718, 467, 55, 76	6204	0,5	6202	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[927, 723, 406, 284, 470, 4	3248	0,6	3215	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[470, 349, 927, 718, 240, 5	9901	0,7	9781	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[723, 834, 927, 52, 457, 89	4791	0,8	4686	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[723, 240, 52, 655, 438, 92	5649	0,9	5566	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[5881, 9543, 8342, 7628, 6	359470	0	359470	TRUE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[7468, 56, 5008, 9810, 457	273694	0,1	273690	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[7468, 7717, 6725, 1816, 6	469211	0,2	468930	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[5509, 8081, 9201, 6714, 9	490821	0,3	490097	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[4859, 2326, 3672, 3793, 6	486611	0,4	486531	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[4859, 8865, 3067, 2659, 7	56063	0,5	55978	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[3837, 1520, 7717, 6575, 4	210577	0,6	210561	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[8591, 2657, 2659, 4021, 8	301924	0,7	301033	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[4574, 6725, 9203, 7628, 7	469260	0,8	468616	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[3136, 1311, 8375, 6615, 7	405048	0,9	404170	FALSE

Figure 7: Randomly chosen set elements in range [0-1000] and set size of [5,20,100] with different error rates



[3497, 5908, 4911, 9357, 6630]	5	[5908, 6630, 9357, 4911]	26806	0	26806	TRUE
[3497, 5908, 4911, 9357, 6630]	5	[3497, 6630, 4911, 5908, 9	30303	0,1	30303	TRUE
[3497, 5908, 4911, 9357, 6630]	5	[3497, 9357, 4911]	17765	0,2	17449	FALSE
[3497, 5908, 4911, 9357, 6630]	5	[6630, 3497, 5908, 9357, 4	30303	0,3	30303	TRUE
[3497, 5908, 4911, 9357, 6630]	5	[5908, 9357, 6630]	21895	0,4	21895	TRUE
[3497, 5908, 4911, 9357, 6630]	5	[4911, 5908, 6630, 3497]	20946	0,5	20176	FALSE
[3497, 5908, 4911, 9357, 6630]	5	[3497, 6630, 9357, 5908, 4	30303	0,6	30303	TRUE
[3497, 5908, 4911, 9357, 6630]	5	[5908, 4911, 6630, 3497]	20946	0,7	20898	FALSE
[3497, 5908, 4911, 9357, 6630]	5	[3497, 6630, 9357]	19484	0,8	19484	TRUE
[3497, 5908, 4911, 9357, 6630]	5	[5908, 9357, 6630]	21895	0,9	19484	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[7061, 7755, 6751]	21567	0	21567	TRUE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[7755, 3779, 5247]	16781	0,1	16762	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[1281, 6031, 8295, 777, 11	54828	0,2	54606	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[8279, 1652, 9360, 1281, 7	60642	0,3	60584	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[6751, 3861, 8279, 3561, 7	36195	0,4	35971	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[8295, 3779, 7061, 777, 60	93690	0,5	92172	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[7061, 1112, 3642, 1281, 7	89272	0,6	87920	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[6461, 9360, 3779, 7755]	27355	0,7	26768	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[7061, 5247, 3642, 3861, 6	35679	0,8	35037	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[7755, 8279]	16034	0,9	15685	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[4021, 2299, 7628, 4831, 8	518328	0	518328	TRUE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[7717, 797, 9065, 5382, 81	316073	0,1	316067	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[2167, 7714, 9397, 6615, 8	523388	0,2	523367	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[4574, 502, 4021, 3185, 76	37145	0,3	37081	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[419, 6615, 7369, 6165, 48	164808	0,4	164488	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[7714, 7112, 3597, 2659, 9	340666	0,5	339926	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[2299, 6615, 1816, 7369, 6	187758	0,6	187448	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[1258, 9914, 2657, 6828, 5	147227	0,7	147216	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[2851, 6615, 1311, 8658]	19435	0,8	19423	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[9376, 797, 9810, 5386, 93	40276	0,9	40224	FALSE

Figure 8: Randomly chosen set elements in range [0-10000] and set size of [5,20,100] with different error rates

5 Discussion

In conclusion, we showed that Subset Sum is NP-complete problem. In order to show that we reduced the known NP-complete problem, 3-SAT problem, to Subset Sum problem. An exact and an efficient algorithm that solves the problem is not known yet. There are some approximation algorithms uses trimming to reduce the number of elements considered for summing. We analyzed an example of that algorithm and we concluded that it is a fully polynomial-time approximation scheme for the Subset Sum problem.

After the experimental analysis of the trimming algorithm, we found that as the set size and number range increases, success rate decreases.When the number range increases, algorithm becomes inefficient even in different error rates.In addition, if set size increases, algorithm becomes more unsuccessful and result is independent from the error rate. When the range is set as 1 to 100 and error rate is 0.1 and set size is 10, success rate is 0.8. Under the same conditions, if we change the set sizes as 20 and 40, success rate becomes 0.6 in both of the set sizes. If we change the error rate as 0.3, our success rates becomes 0.4, 0.30, 0.35 respectively. This result is mainly because the increase in set size leads to increase in the number of trimmed elements and this affects the success rate directly.

Regarding the running time of approximate algorithm in the experimental analysis part, we can observe that the running time is polynomial and set size, number range and error rate don't affect this finding that we obtained from mean exact time calculation.

6 References

1 Cormen, T. (2009). Introduction to algorithms. Cambridge, Mass.: MIT Press.

2 Tardos, E. (2015) Reduction from 3 SAT to Subset Sum. Retrieved from: https://www.cs.cornell.edu/courses/cs4820/2015sp/notes/reduction-subsetsum.pdf