RESEARCH PROJECT OF SUBSET SUM PROBLEM

Final Project

Group Members: Ruoyu Chen, Yang Cui

4/21/2017

Abstract

This paper focused on researching the famous optimization problem: Subset Sum Problem. Subset Sum problem is easy to state and widely applicable. It is NP-Complete. Researchers have studied this problem for many years. In this project, we begin by developing a benchmark for this problem. The benchmark contains several sets of instances each with different size and value. Then varies of optimization techniques/algorithms were developed and implemented for solving instances of the problem and apply them to the benchmark. The algorithms include exhaustive algorithm, greedy algorithm, LP/ILP, Steepest Descent and Simulated Annealing. Results and effectiveness of all the algorithms in this paper will be discussed and compared. Finally, this project helped us learn everything there is to know about the computational solution of subset sum problem.

Contents

Chapter 1

Introduction

This project's topic is Subset Sum problem. Subset Sum is a well-known NP-complete problem. It could be descripted as follows: Given a set of integers and a target value, is there a subset of the integers whose sum equals the target? More detailed description will be present later.

In this project, we will firstly develop a benchmark for the problem. Secondly, we will develop and implement an exhaustive optimal algorithm, which is called backtracking (BT) for solving instances of this problem and apply it to our benchmark. Thirdly, there will be analysis of the result we get.

Problem Description

The problem could be expressed in both decision and optimization version. First, it could be described as a decision problem: Given a set of positive integers S and a target value T, is there any subset of S which has a sum equals to T? It is assumed that the input set S is unique, no duplicates are presented. Second, the optimization version of Subset Sum problem is as follows: Given a set of positive integers S and a target value T, find a largest integer t such that t <= T and there is a subset of S with sum equals to t.

We could also define the algorithm as a language:

$$SUBSET - SUM = \{< S, t > : there\ exists\ a\ subset\ S' \subseteq S\ such\ that\ t = \sum_{s \in S'} s\}$$

Benchmark Description

Represent each value in S by S_i, Target value be T. Problem Size is represented by n. Below table shows how we generate our instances. We have 5 group of instances. The first group of instances was choose from the set {1, 2, 3, …, 100} by taking 10 more numbers each time. For example, the first set is {1, 2, 3, …, 10} and the target is 10, the second set is {1, 2, 3, …, 20} and the target is 20 and so on. So there are 10 instances in this group.

Instance group 2 to 5 are randomly generated. In group 2, 3 and 4, $\boldsymbol{S_i}$is randomly choose from $[1, 10^E]$and target equals to $T = n * \frac{10^E}{4}$ where E equals to 3, 6 and 12. The size of s is 4, 8, 12, ... , 100. In Group 5, we assume $\boldsymbol{S_i}$ are even numbers randomly choose from $[1, 10^3]$. There are total 110 instances in this benchmark. The largest instance is 25 times larger than the smallest instance.

Instance Group	Instance Index	S value S_i	Target (T)	S Size (n)

1	$A_1 - A_{10}$	Input set S is a subset of {1, 2, …, 100}. By choosing 10 more numbers each time.	Target equals to problem size n.	10, 20, 30, …, 100
2	$A_{11} - A_{35}$	$\boldsymbol{S_i}$ randomly choose from $[1, 10^3]$.	$T = n * \frac{10^3}{4}$	4, 8, 12, …, 100
3	$A_{36} - A_{60}$	$\boldsymbol{S_i}$ randomly choose from $[1, 10^6]$.	$T = n * \frac{10^6}{4}$	4, 8, 12, …, 100
4	$A_{61} - A_{85}$	$\boldsymbol{S_i}$ randomly choose from $[1, 10^9]$.	$T = n * \frac{10^9}{4}$	4, 8, 12, …, 100
5	$A_{86} - A_{110}$	Specify $\boldsymbol{S_i}$ even randomly choose from $[1, 10^3]$.	$T = n * \frac{10^3}{4}$	4, 8, 12, …, 100

Table 1 Benchmark description

Exhaustive optimal algorithm

This paper use a recursive algorithm called backtracking (BT) to solving the Subset Sum problem. The algorithm logic is to check every number in the set S. For every element y of S, there has two options, either we will include that element in subset or we don't include it. If there is a subset S' with sum t, it either contains y or it doesn't. If S' contains y, we could add y in the subset, and recursively do this for the next number in S until the subset sum equals t or we have reach the end of S; If S' doesn't contain y, we will skip y, and recursively check the next element.

The pseudo-code of this algorithm is shown as follow:

```
BACKTRACKING (     set S,              // an array of positive integers
                   int currSum,        // current sum of all numbers selected
                   int index,          // the index of current number
                   int sum,            // the target sum
                   set solution)       // an array to contain selected numbers
1.    if currSum == sum
2.          print solution
3.    else if index == S.length
4.          return
5.    else
6.          solution.add(S[index])          // select the element
7.          currSum += S[index];
8.          BACKTRACKING(S, currSum, index + 1, sum, solution);
9.          currSum -= S[index]
10.         solution.remove(S[index])     // unselect the element
11.         recursiveFind(S, currSum, index + 1, sum, solution);
12.   return
```

Figure 1 An exhaustive optimal algorithm for solving instances

Results and Analysis

In our test, we picked up instances from each group, applied our exhaustive algorithm to them and record the elapsed time. There are two cpu time limits: 1 minute and 10 minute. We record on the table when the instance cost more than 1 minute to solve. And when the elapsed time reach to 10 minute also make a record and then stop the test.

There are total 110 tests/instances in 5 groups. We have described all instances in the benchmark description part. Below is the test results and analysis for each group:

Group 1

The results of group 1 instance are shown in the following table:

Instance Group 1	S value Si	Target (T)	S Size (n)	# Subsets	# Recursions	Within 10 minute	Within 1 minute	Time Used (0 if less than 1 second)	Comments
A1	1 - 10	10	10	10	1701	1	1	0	
A2	1 - 20	20	20	64	1864785	1	1	0	
A3	1 - 30	30	30	296	1951227425	1	1	7	
A4	1 - 40	40	40	1113	2.02546E+12	0	0	8079	2.24 hours
A5	1 - 50	50	50	Not be solved in 12 hours.					
A6	1 - 60	60	60						
A7	1 - 70	70	70						
A8	1 - 80	80	80						
A9	1 - 90	90	90						
A10	1 - 100	100	100						

Table 2 Instance group one result

We could see from the result table that 4 out of 10 instances could be solved in relatively short time. When size of array S is smaller than 30, it could be solved very quick, that is, within 10 seconds. However, when the size of S is larger than 30, the elapsed time increased significantly. For example, when the size of S is 40, it consumes 2.24 hours to solved the

instance. And when the instance size is larger than 40, we could not get the result even in 12 hours.

The graph below shows the elapsed time with different sizes.

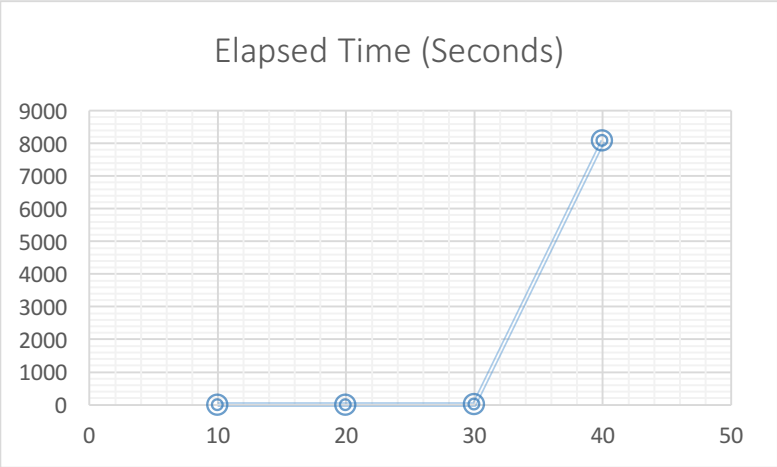

Figure 2 Time complexity of the exhaustive algorithm in group one instances

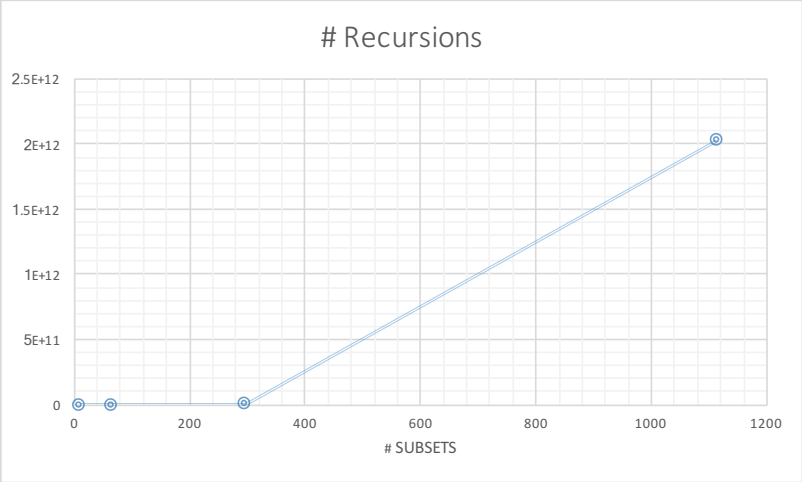

Figure 3 The relationship between number of recursions and subsets

Figure 3 shows the relationship between number of subsets in the solution and how many recursions need to find these solutions. We could see that the trend is similar to the time complexity chart.

Group 2

The result of group 2 instance is shown in below table:

Instance Group 2	S value Si	Target (T)	S Size (n)	# Subsets	# Recursions	Within 10 minute	Within 1 minute	Time Used (0 if less than 1 second)	Comments
A1	random(1, 10^3)	1000	4	0	31	1	1	0	
A2	random(1, 10^3)	2000	8	0	511	1	1	0	
A3	random(1, 10^3)	3000	12	1	8177	1	1	0	
A4	random(1, 10^3)	4000	16	19	131035	1	1	0	
A5	random(1, 10^3)	5000	20	321	2095799	1	1	0	
A6	random(1, 10^3)	6000	24	4535	33513555	1	1	0	
A7	random(1, 10^3)	7000	28	61747	536238635	1	1	2	
A8	random(1, 10^3)	8000	32	1057685	8583864411	1	1	33	
A9	random(1, 10^3)	9000	36	16018212	1.37301E+11	1	0	532	
A10	random(1, 10^3)	10000	40	203555890	2.19636E+12	0	0	8904	
A11	random(1, 10^3)	11000	44						
A12	random(1, 10^3)	12000	48						
A13	random(1, 10^3)	13000	52						
A14	random(1, 10^3)	14000	56						
A15	random(1, 10^3)	15000	60						
A16	random(1, 10^3)	16000	64						
A17	random(1, 10^3)	17000	68						
A18	random(1, 10^3)	18000	72		Not be solved in 12 hours.				
A19	random(1, 10^3)	19000	76						
A20	random(1, 10^3)	20000	80						
A21	random(1, 10^3)	21000	84						
A22	random(1, 10^3)	22000	88						
A23	random(1, 10^3)	23000	92						
A24	random(1, 10^3)	24000	96						
A25	random(1, 10^3)	25000	100						

Table 3 Instance group 2 result

In 12 hours, 10 out of 25 test cases are finished. Given S value S_i be random in 1 to 1000, the largest size that could be solved in 1 minutes is 32 and in 10 minute is 36. That is to say, for this group of instance, 36% of the instances could be solved in 10 minute.

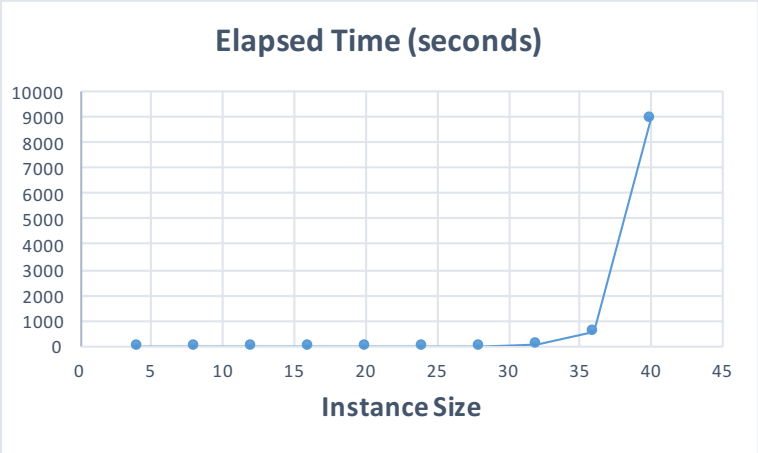

Figure 4 Time complexity of group 2 instances

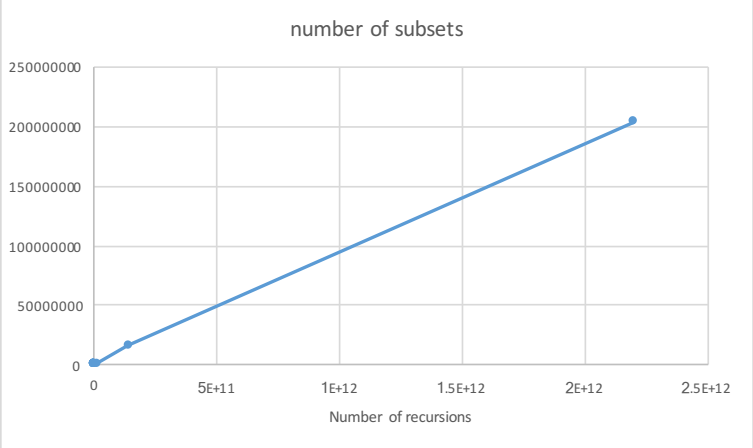

Figure 5 The relationship between #recursions and #subsets

We could see from the above two figures that the time complexity of this algorithm for these instances is exponential. With growing of the instance size, the elapsed time grows significantly. And number of subsets is nearly linearly growth with number of recursions.

Group 3

We got the results for group 3 instances as shown in the following table:

Instance Group 3	S value Si	Target (T)	S Size (n)	# Subsets	# Recursions	Within 10 minute	Within 1 minute	Time Used (0 if less than 1 second)
A1	random(1, 10^6)	1 * 10^6	4	0	31			0
A2	random(1, 10^6)	2 * 10^6	8	0	511			0
A3	random(1, 10^6)	3 * 10^6	12	0	8191			0
A4	random(1, 10^6)	4 * 10^6	16	0	131071			0
A5	random(1, 10^6)	5 * 10^6	20	0	2097151			0
A6	random(1, 10^6)	6 * 10^6	24	2	33554427			0
A7	random(1, 10^6)	7 * 10^6	28	71	536870735			2
A8	random(1, 10^6)	8 * 10^6	32	967	8589931649			34
A9	random(1, 10^6)	9 * 10^6	36	15905	1.37439E+11			557
A10	random(1, 10^6)	10 * 10^6	40	231459	2.19902E+12			8972
A11	random(1, 10^6)	11 * 10^6	44	Not be solved in 12 hours.				
A12	random(1, 10^6)	12 * 10^6	48					
A13	random(1, 10^6)	13 * 10^6	52					
A14	random(1, 10^6)	14 * 10^6	56					
A15	random(1, 10^6)	15 * 10^6	60					
A16	random(1, 10^6)	16 * 10^6	64					
A17	random(1, 10^6)	17 * 10^6	68					
A18	random(1, 10^6)	18 * 10^6	72					
A19	random(1, 10^6)	19 * 10^6	76					
A20	random(1, 10^6)	20 * 10^6	80					
A21	random(1, 10^6)	21 * 10^6	84					
A22	random(1, 10^6)	22 * 10^6	88					
A23	random(1, 10^6)	23 * 10^6	92					
A24	random(1, 10^6)	24 * 10^6	96					
A25	random(1, 10^6)	25 * 10^6	100					

Table 4 Results for group 3 instances

For group 3 instances, 10 out of 25 instances are finished in 12 hours. 36% instances could be solved in 10 minutes and 32% instances were solved in 1 minutes. Compared with group 2, we found that the value in the array is not significantly affect the time to solve the instance. With value be 1000 times larger from group 2 to group 3, we get same amount of instance be solved in 10 minutes. And the time for each solved instance is similar when the size is same.

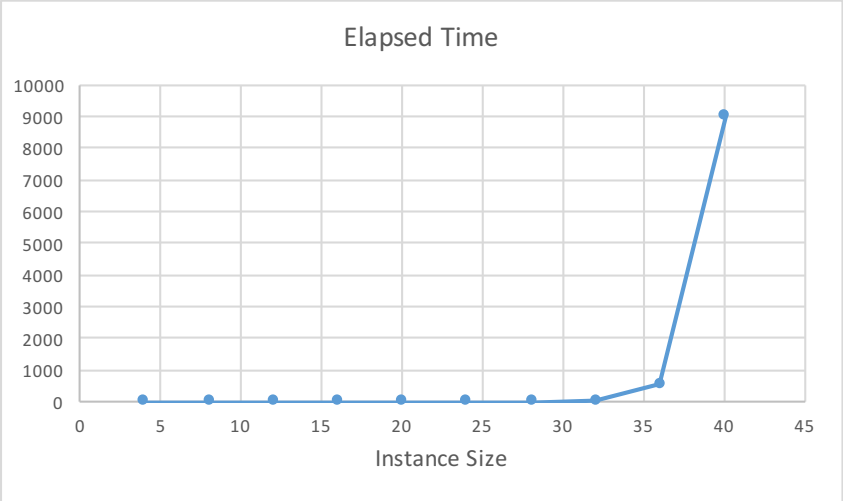

Figure 6 Algorithm time complexity for group 3 instances

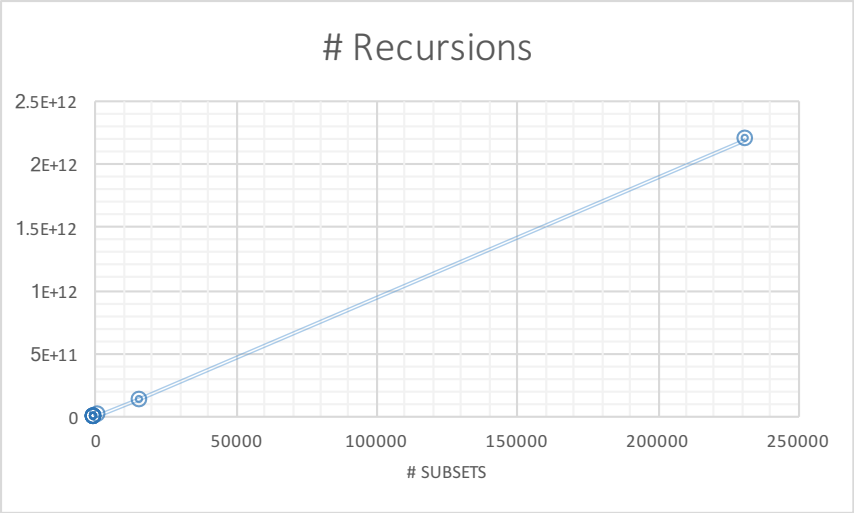

Figure 7 The relationship between #Recursions and #Subsets

Group 4

The result for group 4 instances is show in below table.

Instance Group 4	S value Si	Target (T)	S Size (n)	# Subsets	# Recursions	Within 10 minute	Within 1 minute	Time Used (0 if less than 1 second)
A1	random(1, 10^9)	1 * 10^9	4	0	31			0
A2	random(1, 10^9)	2 * 10^9	8	0	511			0
A3	random(1, 10^9)	3 * 10^9	12	0	8191			0
A4	random(1, 10^9)	4 * 10^9	16	0	131071			0
A5	random(1, 10^9)	5 * 10^9	20	0	2097151			0
A6	random(1, 10^9)	6 * 10^9	24	0	33554431			0
A7	random(1, 10^9)	7 * 10^9	28	0	536870911			2
A8	random(1, 10^9)	8 * 10^9	32	2	8589934591			26
A9	random(1, 10^9)	9 * 10^9	36	17	1.37439E+11			545
A10	random(1, 10^9)	10 * 10^9	40	218	2.19902E+12			8919
A11	random(1, 10^9)	11 * 10^9	44	Not be solved in 12 hours.				
A12	random(1, 10^9)	12 * 10^9	48					
A13	random(1, 10^9)	13 * 10^9	52					
A14	random(1, 10^9)	14 * 10^9	56					
A15	random(1, 10^9)	15 * 10^9	60					
A16	random(1, 10^9)	16 * 10^9	64					
A17	random(1, 10^9)	17 * 10^9	68					
A18	random(1, 10^9)	18 * 10^9	72					
A19	random(1, 10^9)	19 * 10^9	76					
A20	random(1, 10^9)	20 * 10^9	80					
A21	random(1, 10^9)	21 * 10^9	84					
A22	random(1, 10^9)	22 * 10^9	88					
A23	random(1, 10^9)	23 * 10^9	92					
A24	random(1, 10^9)	24 * 10^9	96					
A25	random(1, 10^9)	25 * 10^9	100					

Table 5 Results for group 4 instance

Similar with our conclusion before, the time complexity doesn't change with the value in array S become 1000 times larger. However, the size of S affects the elapsed time very much. For this group test, we solved 10 out of 25 instance in 12 hours. Out of these 25 instances, 36% instances could be solved in 10 minutes and 32% instances were solved in 1 minutes.

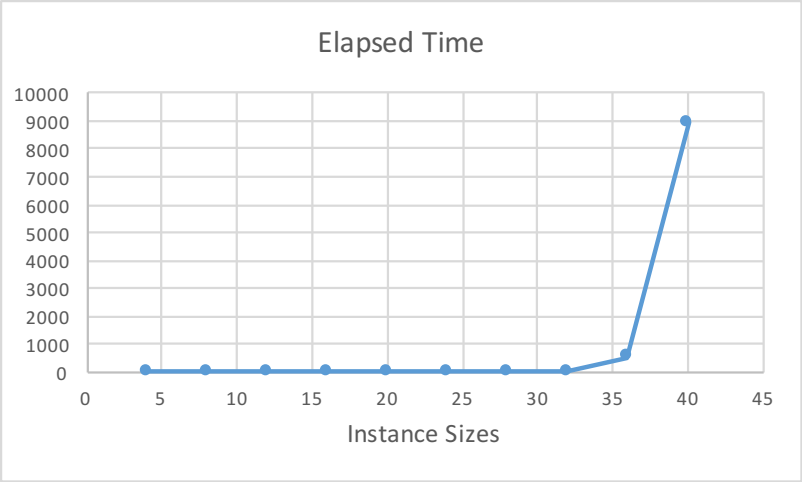

Figure 8 Algorithm time complexity for instance group 4

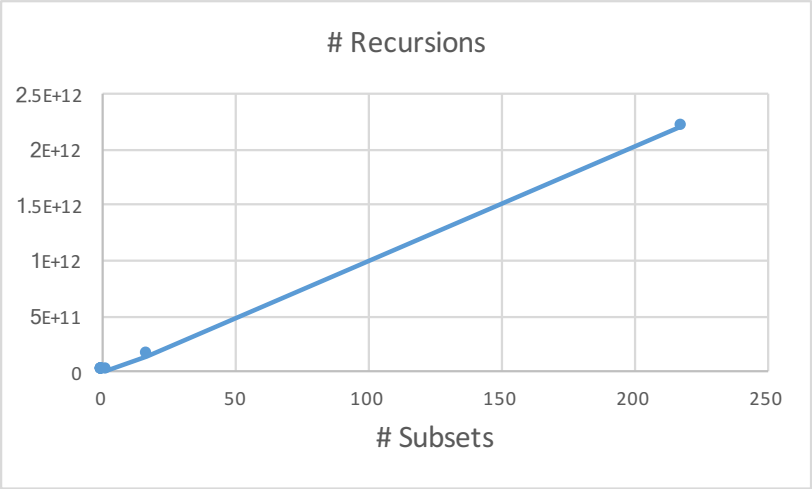

Figure 9 # Recursions vs. # Subsets

Group 5

For group 5 instances, data values are choosing even numbers from 1 to 1000. The result is shown in the following table.

Instance Group 5	S value Si	Target (T)	S Size (n)	# Subsets	# Recursions	Within 10 minute	Within 1 minute	Time Used (0 if less than 1 second)
A1	ndom even (1, 10^	1 * 10^3	4	0	31			0
A2	ndom even (1, 10^	2 * 10^3	8	1	509			0
A3	ndom even (1, 10^	3 * 10^3	12	3	8189			0
A4	ndom even (1, 10^	4 * 10^3	16	43	130947			0
A5	ndom even (1, 10^	5 * 10^3	20	541	2096267			0
A6	ndom even (1, 10^	6 * 10^3	24	5443	33479173			0
A7	ndom even (1, 10^	7 * 10^3	28	145094	536181249			2
A8	ndom even (1, 10^	8 * 10^3	32	1333025	8569380395			34
A9	ndom even (1, 10^	9 * 10^3	36	29928917	1.372E+11			558
A10	ndom even (1, 10^	10 * 10^3	40	472769918	2.19446E+12			9030
A11	ndom even (1, 10^	11 * 10^3	44					
A12	ndom even (1, 10^	12 * 10^3	48					
A13	ndom even (1, 10^	13 * 10^3	52					
A14	ndom even (1, 10^	14 * 10^3	56					
A15	ndom even (1, 10^	15 * 10^3	60					
A16	ndom even (1, 10^	16 * 10^3	64					
A17	ndom even (1, 10^	17 * 10^3	68					
A18	ndom even (1, 10^	18 * 10^3	72		Not be solved in 12 hours.			
A19	ndom even (1, 10^	19 * 10^3	76					
A20	ndom even (1, 10^	20 * 10^3	80					
A21	ndom even (1, 10^	21 * 10^3	84					
A22	ndom even (1, 10^	22 * 10^3	88					
A23	ndom even (1, 10^	23 * 10^3	92					
A24	ndom even (1, 10^	24 * 10^3	96					
A25	ndom even (1, 10^	25 * 10^3	100					

Table 6 Results for group 5 instances

We could see from the above table that 10 out of 25 instances has been solved. 36% instances could be solved in 10 minute and 32% instances were solved in 1 minute. The following charts shows the the algorithm time complexity and relationship between number of subsets and number of recursions.

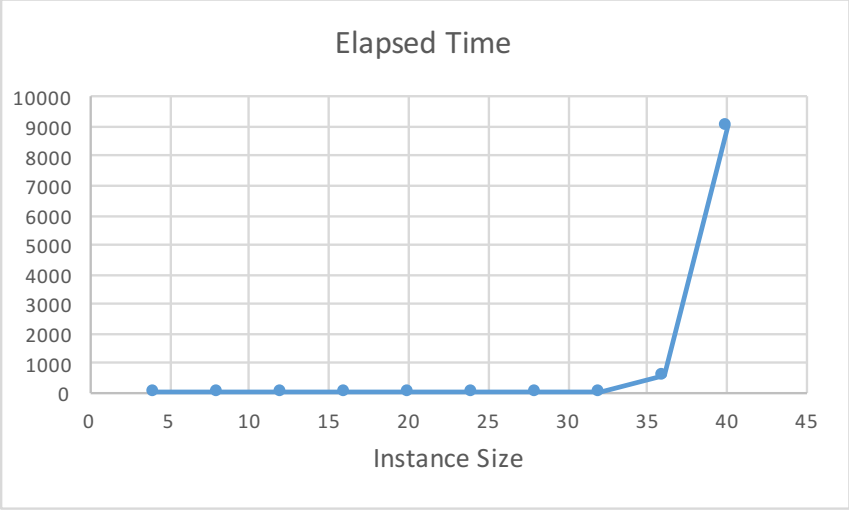

Figure 10 Algorithm time complexity for instance group 5

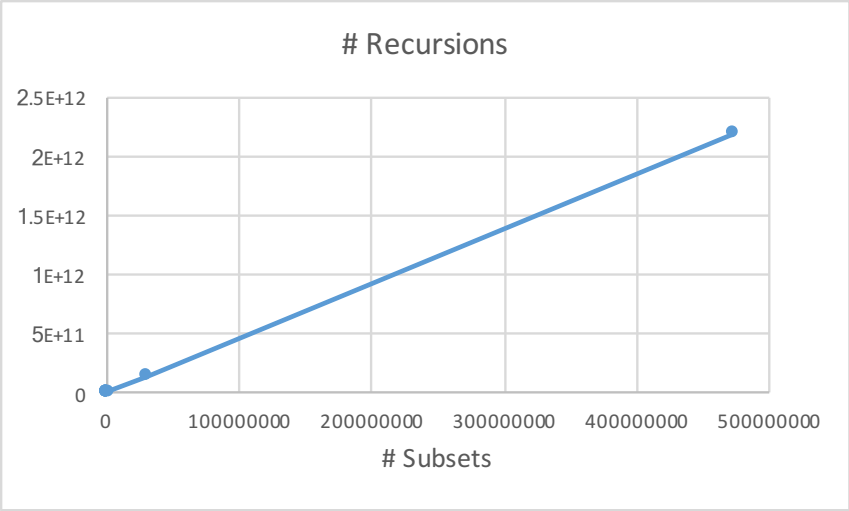

Figure 11 #Recursions vs. #Subsets

Conclusion

Over all 110 instance, 35% of them could be solved by the algorithm within 10 minute. 32% of instance could be solved within 1 minute. The algorithm we use in this experiment is an exhaustive optimal algorithm which will check all subsets of S and then record the subsets which satisfy. It is not an efficient algorithm. This algorithm could only deal well with problem with size less than 40. If the instance size is beyond 40, the time cost for the algorithm to solve the instance will beyond 12 hours.

Chapter 2

Introduction

This is the second part of our research project of Subset Sum problem. In this part, we will survey all complexity results related to the problem. This paper begins with a precise and concise definition of Subset Sum problem in Part II. Then the paper will describe different related problems of Subset Sum in Part III, including NPC sub problem, P sub problem, NPC super problem and other related problems. In part IV, we will describe an algorithm to solve 3SUM problem, which is a sub problem of Subset Sum that is in P.

Problem Definition

The Subset Sum problem is defined as follows: given a set of positive integers S and an integer t, determine whether there is a set S'such that $S' \subseteq S$and $\Sigma(S') = t$.
We could also define the algorithm as a language:

$$SUBSET - SUM = \left\{ < S, t >: there\ exists\ a\ subset\ S' \subseteq S\ such\ that\ t = \sum_{s \in S'} s \right\}$$

Related Problems

NPC Sub problems

Half Sum problem

Definition: given a set of positive integers S and an integer $t = \Sigma(S)/2$, determine whether there is a set S'such that $S' \subseteq S$and $\Sigma(S') = t$.

Half Sum problem is a NPC sub problem of Subset Sum problem. By the definition we know that the Half Sum problem is a special case of Subset Sum problem. For an instance x of Half Sum problem, it is already an instance of Subset Sum problem. x in Half Sum problem is a yes instance if and only if x in Subset Sum problem is a yes instance.

Complexity: Half Sum problem is in NPC. [1]

The relationship between problems and its sub problems is shown in below figure.

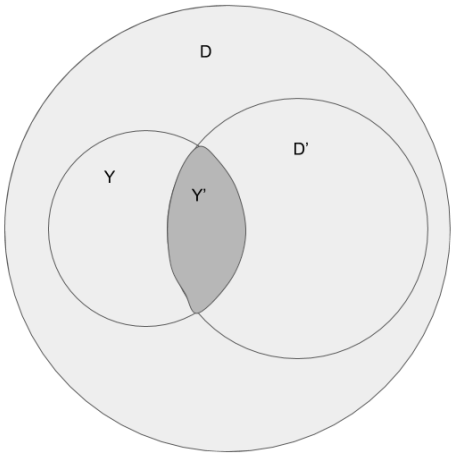

Figure 12 The relationship between a problem and its sub problem

Figure 1 shows relationship between a problem and its sub problem. Consider a problem $\Pi\ (D, Y)$ where $D = set\ of\ all\ instances$ and $Y = set\ of\ all\ yes\ instances$. In a sub problem, we ask the same question over the subset of the domain $\Pi'(D', Y')$, where $D' \subseteq D, Y' \subseteq Y \cap D'$.

Other Sub problems

3SUM problem

General definition: given a set of n real numbers, does it contain three elements that sum to zero?

3SUM problem is a sub problem of Subset Sum problem. We could describe 3SUM problem this way: given a set of positive integers S and an integer $t = 0$, determine whether there is a set S'such that $S' \subseteq S, |S'| = 3$,and $\Sigma(S') = t$. We could see from this definition that given an instance of 3SUM, it is already an instance of Subset Sum. x is a yes instance of 3SUM if and only if x is a yes instance of Subset Sum. 3SUM is a kind of Subset Sum problem over a smaller domain. Thus 3SUM problem is a sub problem of Subset Sum problem.

Complexity: 3SUM problem is in P. It could be solved in Polynomial time. [2]

NPC Super-Problems

Knapsack problem

Definition: given a set of items, each with a weight (w_i) and a value (v_i), determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit W and the total value is larger than or equal to a target value T.

Knapsack problem is a super problem of Subset Sum problem. For every instance in Subset Sum, there is a corresponding Knapsack problem. For example, given an instance of Subset

Sum x: $x = \{< S, t >: there\ exists\ a\ subset\ S' \subseteq S\ such\ that\ t = \sum_{s \in S'} \quad s\}$. We could build a Knapsack problem x' such that it contains same set of items with x. For each item in x', its weight and value is the same and equal to the value of the number in x. The weight bound W and value target T are the same and equal to the target t in x. By this way x is a yes instance if and only if x' is a yes instance. Thus we have proved that Subset Sum problem is a subset of Knapsack problem.

Complexity: Knapsack problem is in NPC. [3]

Below figure shows the relationship of problems described above.

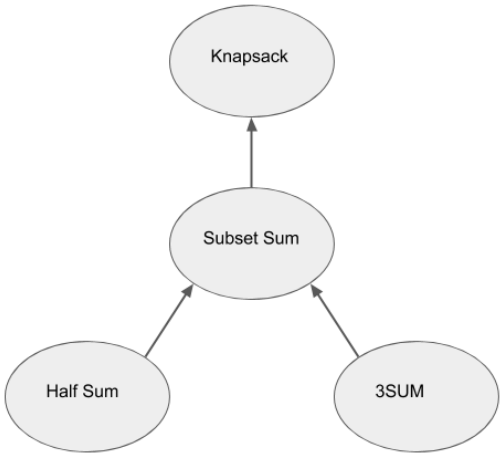

Figure 13 Relationship among problems

Other related problems

Partition Problem

Definition: given a set S of positive integers, the question is whether the numbers can be partitioned into two sets A and $\bar{A} = S - A$ such that $\sum_{x \in A} x = \sum_{x \in \bar{A}} x$.

Complexity: NP-Complete. [6]

Kth largest subset

Definition: given a set A, a size function $s(a) \in Z$ and two non-negative integers B and K, can K distinct subsets of A be found such that the sum of the size of the elements in each subset is less than or equal to B?

Complexity: NP-Hard. [4]

Integer Expression Membership

Definition: Consider arithmetic expressions over the set of N using pair-wise addition (+) and pair-wise multiplication (x) as well as the set operator union (U), intersection (∩), and complement (-). Such formulas are called Integer Expressions. The corresponding membership problem asks whether a given number is in the set computed by the formula.

Complexity: NPC. [5]

Algorithm

3SUM is one of sub problems of Subset Sum that is in P. One of the algorithms that could be used to solve this problem is called quadratic algorithm. Below figure shows the pseudo-code for this algorithm:

```
Quadratic (S)
    sort(S);
    for i = 0 to n - 3 do
        a = S[i]
        start = i + 1
        end = n - 1
        while (start < end) do
            b = S[start]
            c = S[end]
            if (a + b + c == 0) then
                output a, b, c
                end = end - 1
                start = start + 1
            else if (a + b + c > 0) then
                end = end - 1
            else
                start = start + 1
            end
        end
    end
```

Figure 14 Quadratic Algorithm for solving 3SUM problem

The algorithm first sorts the input array and then tests all possible pairs in a careful order that avoids the need to binary search for the pairs in the sorted list, achieving worst-case O(n^2) time.

The correctness of the algorithm can be seen as follows. Suppose we have a solution a + b + c = 0. Since the pointers only move in one direction, we can run the algorithm until the leftmost pointer points to a. Run the algorithm until either one of the remaining pointers points to b or c, whichever occurs first. Then the algorithm will run until the last pointer points to the remaining term, giving the affirmative solution.

Conclusion

This chapter researched complexity results related to our combinatorial optimization problem – Subset Sum problem. Analyzing related problems helps us fully understand the Subset Sum problem. This paper also provided an efficient algorithm for solving 3SUM algorithm. 3SUM is a tractable sub problem of Subset Sum. Efficient solutions to 3SUM could provide important insights into the structure of our problem.

Chapter 3

Introduction

This project will develop and implement a greedy algorithm used for solving instances of Subset Sum problem and evaluate it on the benchmarks. By evaluating the decisions that the algorithm makes, and also compare the results with the exhaustive solutions described in project 1, we will discuss why the Greedy algorithm does not always find the optimal solution for Subset Sum problem.

Greedy Algorithm

The Greedy algorithm we developed is the most natural greedy algorithm. The pseudo code is shown in the following table:

Algorithm	Greedy Algorithm
Input	Set S: An array of positive integers
	int Sum: the target sum
Output	Set A: the heaviest subset that greedy algorithm can find, without exceeding the target value
Greedy	(Set S, int target)
1	Sort(S)
2	A <-- {}
3	while sum(A) < target
4	{
5	find next element e in S in non-increasing order such that sum(A) <= target
6	
7	A <-- {e}
8	}
9	return A

Table 7 Greedy Algorithm for Subset Sum

The greedy algorithm firstly sorts the input number array in non-increasing order. Then it iterates through each element in the array, if the sum of the selected elements array and the new element is less than or equal to the target, it will put the new element into the selected elements array. Or it will continue to check next element till the end of input array.

Results and Analysis

Greedy algorithm is not a good choice to solve Subset Sum problem. This is because greedy algorithm is non-backtracking, which means it will never unselect once it has select one

element. And greedy algorithm always contains the heaviest elements which does not exceed the target limit first. But the optimal solution does not necessarily contain such item. For example, consider an input set S = {7, 6, 3, 1} and a target = 9. We know that the optimal solution is {6, 3}. However, for greedy algorithm, it will first contain the largest element that will not cause a conflict, so {7} will be contained. Then next element will be {1}. Thus greedy algorithm will return {7, 1} as a result. It is not an optimal solution.

Since we have shown that the exhaustive algorithm we developed in project 1 could guarantee to find the optimal solution, and it could find the optimal solution in reasonably time (less than 10 minute) when the instance size is small (less than 40). We could run the greedy algorithm on our benchmark and compare results with the exhaustive algorithm.

The results of greedy algorithm for instance group two are shown in the following table:

Instance Group 2	S value Si	Target (T)	S Size (n)	Within 10 minute	Within 1 minute	Time Used (0 if less than 1 second)	Optimal (1 yes, 0 no)
A1	random(1, 10^3)	1000	4	1	1	0	0
A2	random(1, 10^3)	2000	8	1	1	0	0
A3	random(1, 10^3)	3000	12	1	1	0	0
A4	random(1, 10^3)	4000	16	1	1	0	0
A5	random(1, 10^3)	5000	20	1	1	0	0
A6	random(1, 10^3)	6000	24	1	1	0	0
A7	random(1, 10^3)	7000	28	1	1	0	0
A8	random(1, 10^3)	8000	32	1	1	0	0
A9	random(1, 10^3)	9000	36	1	1	0	0
A10	random(1, 10^3)	10000	40	1	1	0	0
A11	random(1, 10^3)	11000	44	1	1	0	0
A12	random(1, 10^3)	12000	48	1	1	0	0
A13	random(1, 10^3)	13000	52	1	1	0	0
A14	random(1, 10^3)	14000	56	1	1	0	0
A15	random(1, 10^3)	15000	60	1	1	0	0
A16	random(1, 10^3)	16000	64	1	1	0	0
A17	random(1, 10^3)	17000	68	1	1	0	0
A18	random(1, 10^3)	18000	72	1	1	0	0
A19	random(1, 10^3)	19000	76	1	1	0	0
A20	random(1, 10^3)	20000	80	1	1	0	0
A21	random(1, 10^3)	21000	84	1	1	0	0
A22	random(1, 10^3)	22000	88	1	1	0	0
A23	random(1, 10^3)	23000	92	1	1	0	0
A24	random(1, 10^3)	24000	96	1	1	0	0
A25	random(1, 10^3)	25000	100	1	1	0	0

Table 8 Greedy Algorithm Result for Instance Group 2

We could see from above table that the greedy algorithm solved all the instances really quick. However, none of the results returned by the greedy algorithm are optimal solution. Below graph shows the quality of greedy algorithm compared with the exhaustive algorithm for instance size from 4 to 36:

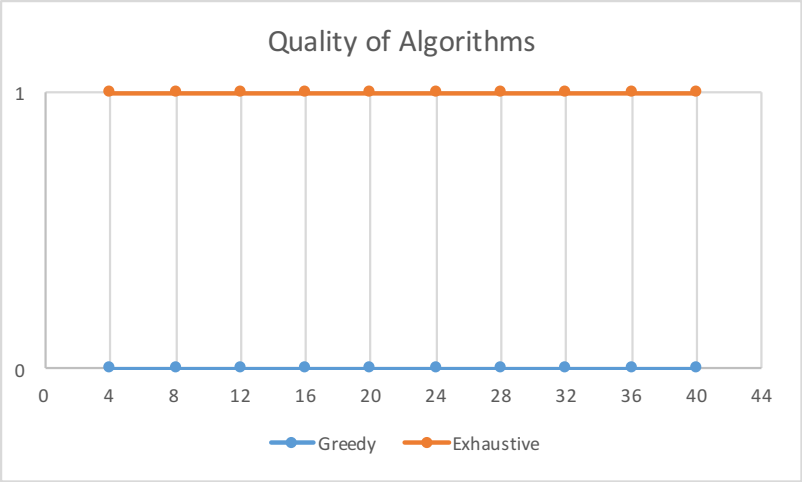

Figure 15 Quality of the solutions as a funciton of instance size

When the instance size is small, exhaustive algorithm guarantee to find an optimal solution while greedy algorithm cannot find any optimal solutions.

Below figure shows the time consumed by each algorithm as a function of instance size:

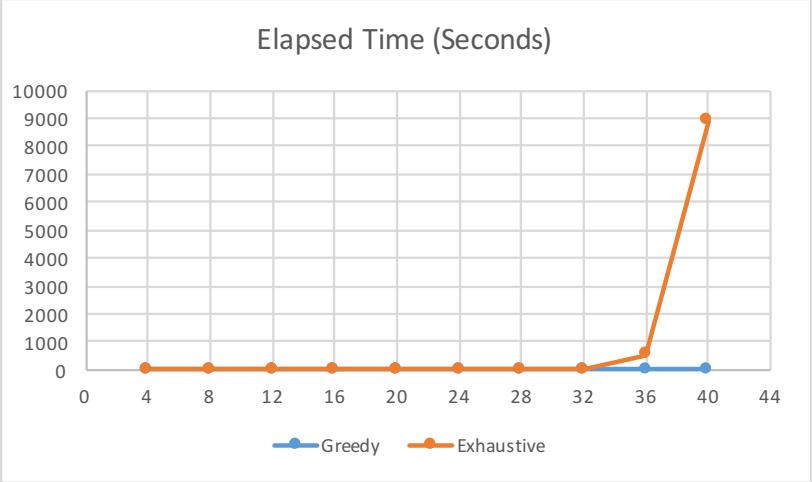

Figure 16 Time complexity of greedy algorithm and exhaustive algorithm

The above figure shows that compared to exhaustive algorithm, the greedy algorithm is very quick. Greedy algorithm runs in polynomial time. For this group of instances, greedy algorithm could get result of all instance sizes within 1 second.

Conclusion

In this chapter we have shown that greedy algorithm is not a good choice to solve Subset Sum problem. Most of time the greedy algorithm couldn't find the optimal solution. We prefer to use exhaustive algorithm since it can guarantee to find the optimal solution.

However, greedy algorithm is very fast. It runs in polynomial time. It could solve very large instance in seconds which may cost exhaustive algorithm days to calculate. Therefore, greedy

algorithm is recommended for an instance of Subset Sum problem when optimal solution is not required, but speed of the algorithm is very needed.

Chapter 4

Introduction

The goal of this phase of the project is to use LP and ILP techniques to attempt to solve instances of Subset Sum problem. We begin by developing an ILP formulation of the problem. Then Using the LP relaxation of these models, compute lower bounds on the objective function for the instances in our benchmark.

ILP formulations will be used to solve every instances in all 5 groups of our benchmark. For each result of an instance, we evaluate the quality of the greedy solution relative to the optimal value, and evaluate the quality of the LP lower bound relative to the optimal value.

Besides, this paper will also compare the effectiveness of ILP formulation with exhaustive algorithm.

ILP formulation for Subset Sum

The Integer Linear Programming (ILP) model for Subset Sum problem is described as follows:

$$minimize\ z = \ T - \sum_{j=1}^{n} s_j * x_j$$

$$subject\ to \sum_{j=1}^{n} s_j * x_j \leq T$$

$$x_j \in \{0, 1\}, j \in \{1, \dots, n\}$$

LP lower bounds

The bound B for an instance could be defined as the difference between target T and the sum of result returned by one algorithm. For our subset sum problem, the optimal lower bound for every instance is 0 since we are trying to find one subset that sums to T.

$$B_{OPT} = 0$$

LP lower bounds could be calculated by removing integer constraints in ILP model. For our Subset Sum problem instances, if let $x_j \in [0, 1]$ instead of {0, 1}, there always exist set x such that $\sum_{j=1}^{n} s_j * x_j = T$. Therefore, LP lower bound for Subset Sum problem equals to the optimal bound, and could be denote as follows:

$$B_{LP} = 0$$

Results and analysis

Data representation

For all our instances, we not only compute exact solutions, but also evaluate the gaps between the lower bound and exact solution, and the gaps between the exact solution and the greedy solution.

Since the size and data value of differences in our benchmark varies significantly, it is in appropriate to use the absolute gap value between the results and the bound. Alternatively, we will use the relative value.

We define “ratio bound” R as the ratio of calculated bound of an instance and the corresponding target. Bigger ratio bound means that this bound/result returned by an algorithm is farther away from the optimal value. For an example, since LP lower bound is the optimal bound for our instances, the ratio bound $R_{LP} = \frac{B_{LP}}{T} = 0$. The gaps between the exact solution and the greedy solution could be expressed as the difference between R_{GREEDY} and R_{ILP}.

We define gap G as the difference between ILP ratio bound and greedy ratio bound $G = R_{GREEDY} - R_{ILP}$. In our practice, we found that the result of ILP is always better than the result of greedy, thus G is always greater than zero. Larger G means ILP performs better than greedy algorithm for an instance.

Results and Analysis

We computed exact solutions for all instances in our benchmark using ILP. We will compare the effectiveness of the ILP formulation with both the greedy algorithm and the exhaustive algorithm we developed in the past projects. The detailed results are shown as follows:

Group 1

The results of group 1 instances are shown in the following table:

				Exhausitive Algorithm (1 minutes limit)			Exhausitive Algorithm (5 minutes limit)			ILP Results			Greedy Algorithm			
Instance Group 1	S value Si	Target (T)	S Size (n)	Optimal?	B	R	Optimal?	B	R	ILP_Optimal?	B_ILP	R_ILP	GREEDY_Optima	B_GREEDY	R_GREEDY	G
A1	1 - 10	1.00E+01	10	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%
A2	1 - 20	2.00E+01	20	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%
A3	1 - 30	3.00E+01	30	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%
A4	1 - 40	4.00E+01	40	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%
A5	1 - 50	5.00E+01	50	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%
A6	1 - 60	6.00E+01	60	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%
A7	1 - 70	7.00E+01	70	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%
A8	1 - 80	8.00E+01	80	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%
A9	1 - 90	9.00E+01	90	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%
A10	1 - 100	1.00E+02	100	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%

Table 9 Group 1 instances results

We could see from results that group 1 instance is very easy to solve for ILP, greedy and exhaustive algorithms. All 10 instances in this group have been optimally solved by every algorithm. Since the result for this group of instance is very straightforward to see. No more graphs will be draw for this group to demonstrate result.

Group 2

Group 2 instances result are as follows:

				Exhausitive (1 minute)			Exhausitive (5 minutes)			ILP			Greedy			
Instance Group 2	**S value Si**	**Target (T)**	**S Size (n)**	**Optimal?**	**B**	**R**	**Optimal?**	**B**	**R**	**ILP_Optimal?**	**B_ILP**	**R_ILP**	**GREEDY_Optimal?**	**B_GREEDY**	**R_GREEDY**	**G**
A11	random(1, 10^3)	1.00E+03	4	FALSE	212	21.2%	FALSE	212	21.2%	FALSE	212	21.2%	TRUE	212	21.2%	0.0%
A12	random(1, 10^3)	2.00E+03	8	FALSE	9	0.5%	FALSE	9	0.5%	FALSE	9	0.5%	FALSE	32	1.6%	1.2%
A13	random(1, 10^3)	3.00E+03	12	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	142	4.7%	4.7%
A14	random(1, 10^3)	4.00E+03	16	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	14	0.4%	0.4%
A15	random(1, 10^3)	5.00E+03	20	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	16	0.3%	0.3%
A16	random(1, 10^3)	6.00E+03	24	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%
A17	random(1, 10^3)	7.00E+03	28	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	9	0.1%	0.1%
A18	random(1, 10^3)	8.00E+03	32	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	3	0.0%	0.0%
A19	random(1, 10^3)	9.00E+03	36	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%
A20	random(1, 10^3)	1.00E+04	40	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	3	0.0%	0.0%
A21	random(1, 10^3)	1.10E+04	44	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	14	0.1%	0.1%
A22	random(1, 10^3)	1.20E+04	48	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	5	0.0%	0.0%
A23	random(1, 10^3)	1.30E+04	52	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	2	0.0%	0.0%
A24	random(1, 10^3)	1.40E+04	56	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	3	0.0%	0.0%
A25	random(1, 10^3)	1.50E+04	60	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	10	0.1%	0.1%
A26	random(1, 10^3)	1.60E+04	64	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	2	0.0%	0.0%
A27	random(1, 10^3)	1.70E+04	68	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	6	0.0%	0.0%
A28	random(1, 10^3)	1.80E+04	72	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	14	0.1%	0.1%
A29	random(1, 10^3)	1.90E+04	76	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	1	0.0%	0.0%
A30	random(1, 10^3)	2.00E+04	80	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	5	0.0%	0.0%
A31	random(1, 10^3)	2.10E+04	84	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	12	0.1%	0.1%
A32	random(1, 10^3)	2.20E+04	88	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%
A33	random(1, 10^3)	2.30E+04	92	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	6	0.0%	0.0%
A34	random(1, 10^3)	2.40E+04	96	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	21	0.1%	0.1%
A35	random(1, 10^3)	2.50E+04	100	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%

Table 10 Group 2 instance results

In this group of instances, we could see that all the exact solutions using ILP have been found. Most of the solutions meet the lower bound (gaps are 0). Exact solution for instance A11 is 21.2% from the lower bound, and A12 is 0.5% from the lower bound.

The following graph shows the ratio bound of ILP results and greedy results compared to optimal/LP results.

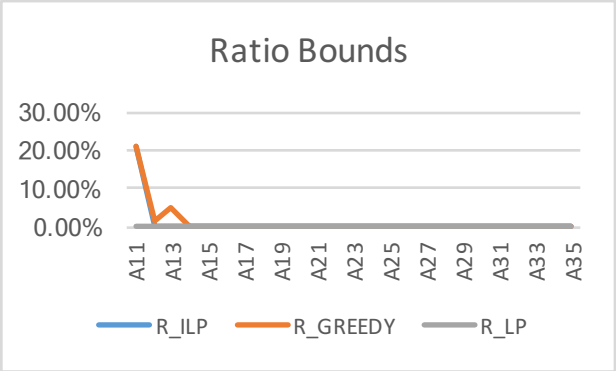

Figure 17 Ratio bounds for instance group 2

We could see that for smaller size instance, ILP and greedy both preform not very well. While when the size of instances gets larger and larger, their performance gets nearer and nearer to the optimal bound.

This figure shows the gap between exact solutions and greedy solutions.

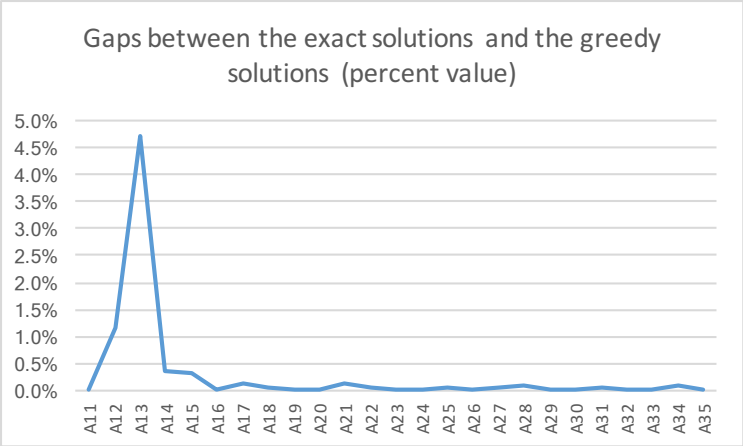

Figure 18 Gaps between exact solutions and the greedy solutions for instance group 2

We could see that the performance of ILP formulas are always better than greedy algorithm. When the instance size getting larger, their performance gets similar, that is also means that the LP bound gets tighter.

The following graph shows the the effectiveness of ILP formulation with the exhaustive algorithm.

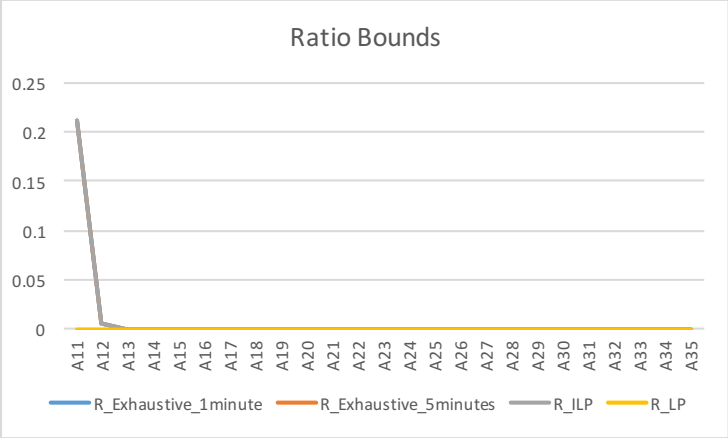

Figure 19 Ratio Bounds

For this group of instance, the results show that exhaustive algorithm and ILP algorithm has same performance. Since both algorithms could guarantee to find an optimal solution given that the instance size is small enough.

Group 3

The following table shows exact solutions using ILP for group 3 data and also the performance (bound) for each instance. Exhaustive and greedy results are also in this table.

				Exhausitive (1 minute)			Exhausitive (5 minutes)			ILP			Greedy				
Instance Group 3	S value Si	Target (T)	S Size (n)	Optimal?	B	R	Optimal?	B	R	ILP_Optimal?	B_ILP	R_ILP	GREEDY_Optimal?	B_GREEDY	R_GREEDY	G	R_LP
A36	random(1, 10^6)	1.00E+06	4	FALSE	18656	1.9%	FALSE	18656	1.9%	FALSE	18656	1.9%	FALSE	75795	7.6%	5.7%	0.0%
A37	random(1, 10^6)	2.00E+06	8	FALSE	16660	0.8%	FALSE	16660	0.8%	FALSE	16660	0.8%	FALSE	22716	1.1%	0.3%	0.0%
A38	random(1, 10^6)	3.00E+06	12	FALSE	1189	0.0%	FALSE	1189	0.0%	FALSE	1189	0.0%	FALSE	82609	2.8%	2.7%	0.0%
A39	random(1, 10^6)	4.00E+06	16	FALSE	64	0.0%	FALSE	64	0.0%	FALSE	64	0.0%	FALSE	42803	1.1%	1.1%	0.0%
A40	random(1, 10^6)	5.00E+06	20	FALSE	2	0.0%	FALSE	2	0.0%	FALSE	2	0.0%	FALSE	7124	0.1%	0.1%	0.0%
A41	random(1, 10^6)	6.00E+06	24	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	1474	0.0%	0.0%	0.0%
A42	random(1, 10^6)	7.00E+06	28	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	28314	0.4%	0.4%	0.0%
A43	random(1, 10^6)	8.00E+06	32	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	63587	0.8%	0.8%	0.0%
A44	random(1, 10^6)	9.00E+06	36	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	9054	0.1%	0.1%	0.0%
A45	random(1, 10^6)	1.00E+07	40	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	4369	0.0%	0.0%	0.0%
A46	random(1, 10^6)	1.10E+07	44	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	15658	0.1%	0.1%	0.0%
A47	random(1, 10^6)	1.20E+07	48	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	26522	0.2%	0.2%	0.0%
A48	random(1, 10^6)	1.30E+07	52	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	708	0.0%	0.0%	0.0%
A49	random(1, 10^6)	1.40E+07	56	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	3486	0.0%	0.0%	0.0%
A50	random(1, 10^6)	1.50E+07	60	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	4513	0.0%	0.0%	0.0%
A51	random(1, 10^6)	1.60E+07	64	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	856	0.0%	0.0%	0.0%
A52	random(1, 10^6)	1.70E+07	68	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	3808	0.0%	0.0%	0.0%
A53	random(1, 10^6)	1.80E+07	72	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	134	0.0%	0.0%	0.0%
A54	random(1, 10^6)	1.90E+07	76	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	1613	0.0%	0.0%	0.0%
A55	random(1, 10^6)	2.00E+07	80	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	163	0.0%	0.0%	0.0%
A56	random(1, 10^6)	2.10E+07	84	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	20386	0.1%	0.1%	0.0%
A57	random(1, 10^6)	2.20E+07	88	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	5606	0.0%	0.0%	0.0%
A58	random(1, 10^6)	2.30E+07	92	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	9259	0.0%	0.0%	0.0%
A59	random(1, 10^6)	2.40E+07	96	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	11423	0.0%	0.0%	0.0%
A60	random(1, 10^6)	2.50E+07	100	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	7272	0.0%	0.0%	0.0%

Table 11 Group 3 instances results

For group 3 instances, ILP and greedy both works better than the previous group. Ratio bound R_ILP is at most 1.9%, it means the LP bound is very tight. The results returned by ILP algorithm and greedy algorithm are very close to or equal to the optimal value.

The following graphs show the gaps between the lower bound and exact solution, and the gaps between the exact solution and the greedy solution.

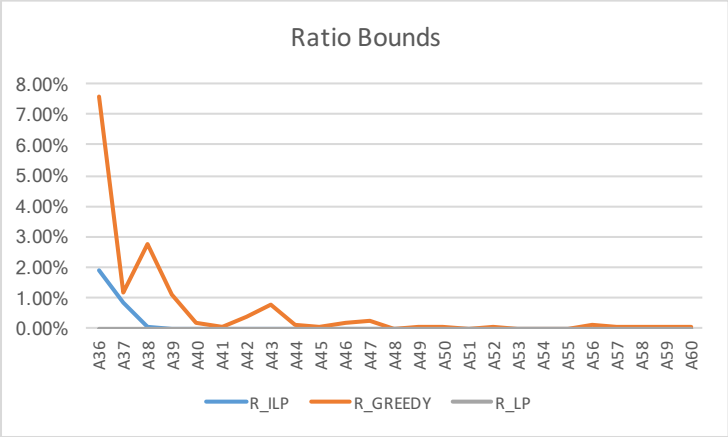

Figure 20 Ratio bounds for instance group 3

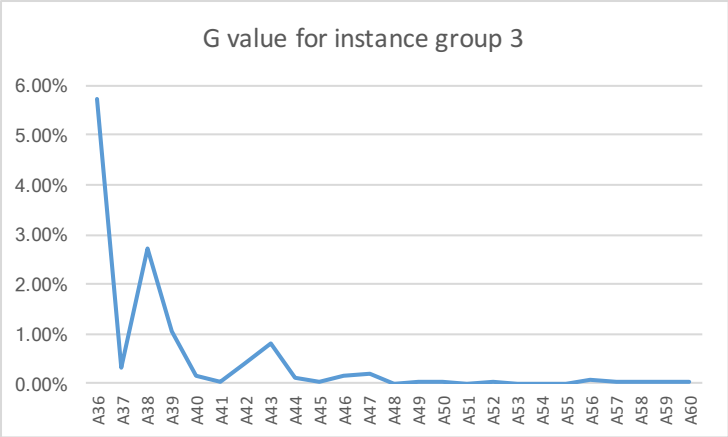

Figure 21 G value for instance group 3

For this group of instance, LP lower bound is tighter than group 2. Both performance become better and similar with the increase of instance size.

The following graph shows the the effectiveness of ILP formulation with the exhaustive algorithm.

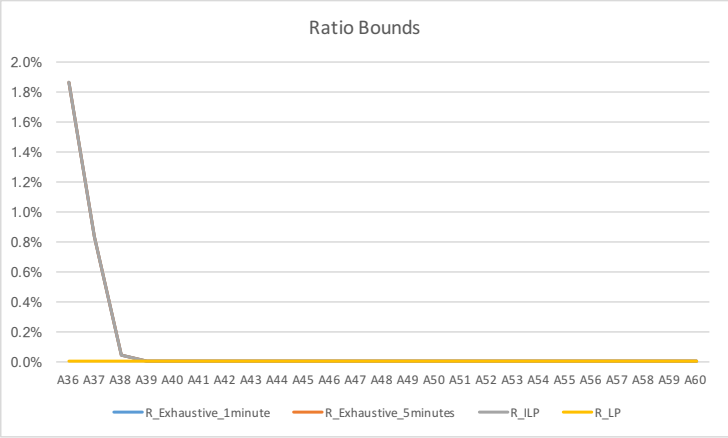

Figure 22 Ratio bounds for instance group 3

Group 4

Below table shows results for instance group 4. This table record results for three algorithms: exhaustive, greedy and ILP.

				Exhausitive (1 minute)			Exhausitive (5 minutes)			ILP			Greedy				
Instance Group 4	S value Si	Target (T)	S Size (n)	Optimal?	B	R	Optimal?	B	R	ILP_Optimal?	B_ILP	R_ILP	GREEDY_Optimal?	B_GREEDY	R_GREEDY	G	R_LP
A61	random(1, 10^9)	1.00E+09	4	FALSE	4E+07	3.8%	FALSE	38388010	3.8%	FALSE	38388010	3.8%	FALSE	38388010	3.8%	0.0%	0.0%
A62	random(1, 10^9)	2.00E+09	8	FALSE	2E+07	1.2%	FALSE	24696486	1.2%	FALSE	24696486	1.2%	FALSE	24696486	1.2%	0.0%	0.0%
A63	random(1, 10^9)	3.00E+09	12	FALSE	3E+05	0.0%	FALSE	270359	0.0%	FALSE	270359	0.0%	FALSE	74446731	2.5%	2.5%	0.0%
A64	random(1, 10^9)	4.00E+09	16	FALSE	6923	0.0%	FALSE	6923	0.0%	FALSE	6923	0.0%	FALSE	64135615	1.6%	1.6%	0.0%
A65	random(1, 10^9)	5.00E+09	20	FALSE	92	0.0%	FALSE	92	0.0%	FALSE	92	0.0%	FALSE	10632307	0.2%	0.2%	0.0%
A66	random(1, 10^9)	6.00E+09	24	FALSE	359	0.0%	FALSE	359	0.0%	TRUE	0	0.0%	FALSE	42380253	0.7%	0.7%	0.0%
A67	random(1, 10^9)	7.00E+09	28	FALSE	59	0.0%	FALSE	59	0.0%	TRUE	0	0.0%	FALSE	21235486	0.3%	0.3%	0.0%
A68	random(1, 10^9)	8.00E+09	32	FALSE	8	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	13810429	0.2%	0.2%	0.0%
A69	random(1, 10^9)	9.00E+09	36	FALSE	1	0.0%	FALSE	1	0.0%	TRUE	0	0.0%	FALSE	4143095	0.0%	0.0%	0.0%
A70	random(1, 10^9)	1.00E+10	40	FALSE	35	0.0%	FALSE	1	0.0%	TRUE	0	0.0%	FALSE	6056313	0.1%	0.1%	0.0%
A71	random(1, 10^9)	1.10E+10	44	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	3691540	0.0%	0.0%	0.0%
A72	random(1, 10^9)	1.20E+10	48	FALSE	25	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	1762326	0.0%	0.0%	0.0%
A73	random(1, 10^9)	1.30E+10	52	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	1832717	0.0%	0.0%	0.0%
A74	random(1, 10^9)	1.40E+10	56	FALSE	3	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	5954176	0.0%	0.0%	0.0%
A75	random(1, 10^9)	1.50E+10	60	FALSE	20	0.0%	FALSE	4	0.0%	TRUE	0	0.0%	FALSE	18049950	0.1%	0.1%	0.0%
A76	random(1, 10^9)	1.60E+10	64	FALSE	2	0.0%	FALSE	1	0.0%	TRUE	0	0.0%	FALSE	12682461	0.1%	0.1%	0.0%
A77	random(1, 10^9)	1.70E+10	68	FALSE	1	0.0%	FALSE	1	0.0%	TRUE	0	0.0%	FALSE	8883659	0.1%	0.1%	0.0%
A78	random(1, 10^9)	1.80E+10	72	FALSE	3	0.0%	FALSE	1	0.0%	TRUE	0	0.0%	FALSE	9585820	0.1%	0.1%	0.0%
A79	random(1, 10^9)	1.90E+10	76	FALSE	66	0.0%	FALSE	1	0.0%	TRUE	0	0.0%	FALSE	1811588	0.0%	0.0%	0.0%
A80	random(1, 10^9)	2.00E+10	80	FALSE	12	0.0%	FALSE	1	0.0%	TRUE	0	0.0%	FALSE	3993447	0.0%	0.0%	0.0%
A81	random(1, 10^9)	2.10E+10	84	FALSE	2	0.0%	FALSE	2	0.0%	TRUE	0	0.0%	FALSE	4345837	0.0%	0.0%	0.0%
A82	random(1, 10^9)	2.20E+10	88	FALSE	8	0.0%	FALSE	3	0.0%	TRUE	0	0.0%	FALSE	2701694	0.0%	0.0%	0.0%
A83	random(1, 10^9)	2.30E+10	92	FALSE	5	0.0%	FALSE	5	0.0%	TRUE	0	0.0%	FALSE	4382356	0.0%	0.0%	0.0%
A84	random(1, 10^9)	2.40E+10	96	FALSE	5	0.0%	FALSE	3	0.0%	TRUE	0	0.0%	FALSE	4176600	0.0%	0.0%	0.0%
A85	random(1, 10^9)	2.50E+10	100	FALSE	13	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	23544824	0.1%	0.1%	0.0%

Table 12 Results for instance group 4

In this group of instances, greedy algorithm performs almost the same with ILP. Their ratio bounds are both at most 3.8%, the LP lower bounds are also very tight. With instance value be 1000 times larger than group 3, we didn't see much change on the performance.

The following pictures show the gaps between the lower bound and exact solution, and the gaps between the exact solution and the greedy solution.

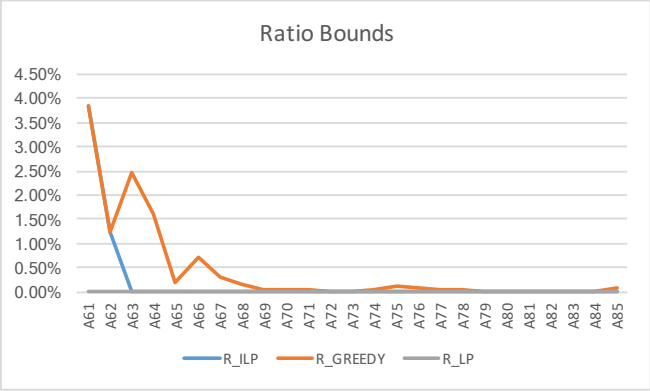

Figure 23 Ratio bounds for instance group 4

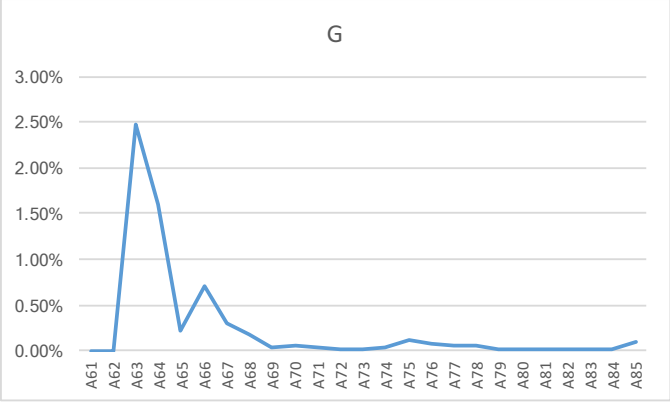

Figure 24 G values for instance group 4

The following graph shows the the effectiveness of ILP formulation with the exhaustive algorithm.

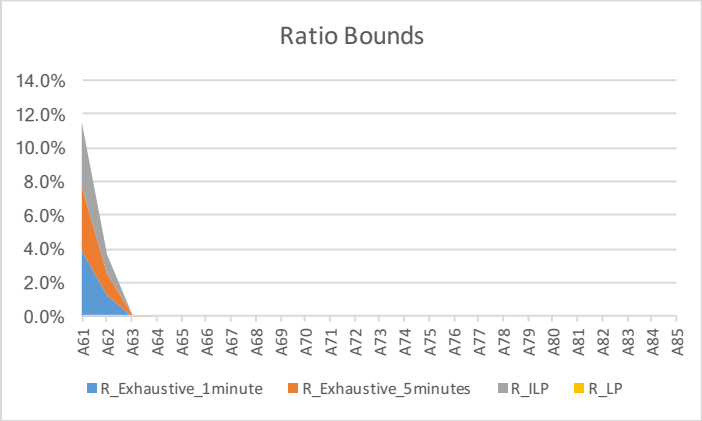

Figure 25 Ratio bounds for instance group 4

We could see from the graph that if given more time, exhaustive algorithm could have better performance for some instance, such as A68, A70, A72 etc. However, for many other smaller instances, exhaustive algorithm could find optimal solution within one minute, so it is no difference for these instances whether the time limit is 1 minute or 5 minute. For ILP formulas, most instance could be solved optimally within 1 minute.

Group 5

The group 5 instances results are shown in the following graph:

				Exhaustive (1 minute)			Exhaustive (5 minutes)			ILP			Greedy				
Instance Group 5	**S value Si**	**Target (T)**	**S Size (n)**	**Optimal?**	**B**	**R**	**Optimal?**	**B**	**R**	**ILP_Optimal?**	**B_ILP**	**R_ILP**	**GREEDY_Optimal?**	**B_GREEDY**	**R_GREEDY**	**G**	**R_LP**
A86	random even (1, 10^3)	1.00E+03	4	FALSE	182	18.2%	FALSE	182	18.2%	FALSE	182	18.2%	FALSE	182	18.2%	0.0%	0.0%
A87	random even (1, 10^3)	2.00E+03	8	FALSE	2	0.1%	FALSE	2	0.1%	FALSE	2	0.1%	FALSE	38	1.9%	1.8%	0.0%
A88	random even (1, 10^3)	3.00E+03	12	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	4	0.1%	0.1%	0.0%
A89	random even (1, 10^3)	4.00E+03	16	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%	0.0%
A90	random even (1, 10^3)	5.00E+03	20	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	60	1.2%	1.2%	0.0%
A91	random even (1, 10^3)	6.00E+03	24	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	4	0.1%	0.1%	0.0%
A92	random even (1, 10^3)	7.00E+03	28	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	12	0.2%	0.2%	0.0%
A93	random even (1, 10^3)	8.00E+03	32	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	24	0.3%	0.3%	0.0%
A94	random even (1, 10^3)	9.00E+03	36	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	14	0.2%	0.2%	0.0%
A95	random even (1, 10^3)	1.00E+04	40	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	38	0.4%	0.4%	0.0%
A96	random even (1, 10^3)	1.10E+04	44	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%	0.0%
A97	random even (1, 10^3)	1.20E+04	48	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	2	0.0%	0.0%	0.0%
A98	random even (1, 10^3)	1.30E+04	52	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%	0.0%
A99	random even (1, 10^3)	1.40E+04	56	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	4	0.0%	0.0%	0.0%
A100	random even (1, 10^3)	1.50E+04	60	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	8	0.1%	0.1%	0.0%
A101	random even (1, 10^3)	1.60E+04	64	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	4	0.0%	0.0%	0.0%
A102	random even (1, 10^3)	1.70E+04	68	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	2	0.0%	0.0%	0.0%
A103	random even (1, 10^3)	1.80E+04	72	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	10	0.1%	0.1%	0.0%
A104	random even (1, 10^3)	1.90E+04	76	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	18	0.1%	0.1%	0.0%
A105	random even (1, 10^3)	2.00E+04	80	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	18	0.1%	0.1%	0.0%
A106	random even (1, 10^3)	2.10E+04	84	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	2	0.0%	0.0%	0.0%
A107	random even (1, 10^3)	2.20E+04	88	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%	0.0%
A108	random even (1, 10^3)	2.30E+04	92	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	FALSE	6	0.0%	0.0%	0.0%
A109	random even (1, 10^3)	2.40E+04	96	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%	0.0%
A110	random even (1, 10^3)	2.50E+04	100	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	TRUE	0	0.0%	0.0%	0.0%

Table 13 Results for instance group 5

For group 5 instances, ILP perform very well, 23/25 instances have been optimally solved. Ratio bounds for most instances are 0%. For greedy algorithm, though only 6 out of 25 instances have been optimally solved, the results for other instances are very close to LP bound.

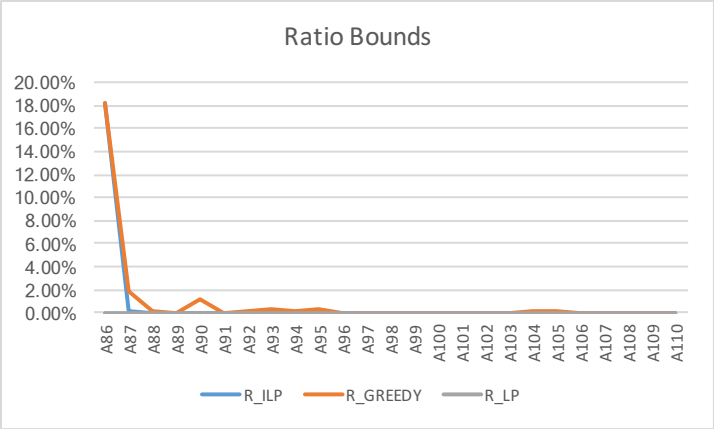

Figure 26 Ratio bounds for instance group 5

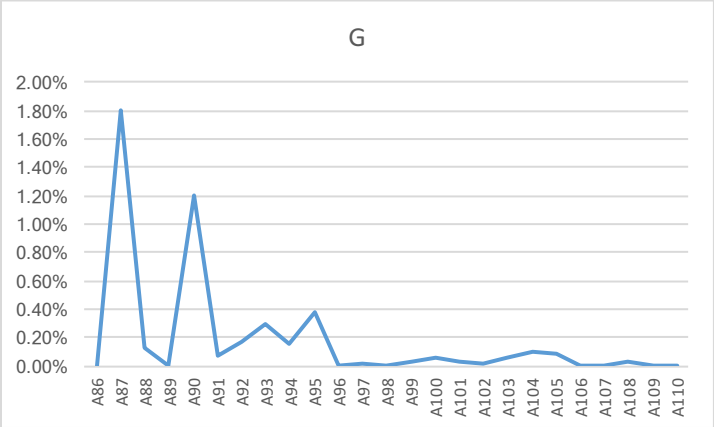

Figure 27 G values for instance group 5

We could see from the above pictures that the ILP bound is very close to greedy bound. With the size of instance become larger, their bounds both tends to the optimal lower bound.

The following graph shows the the effectiveness of ILP formulation with the exhaustive algorithm.

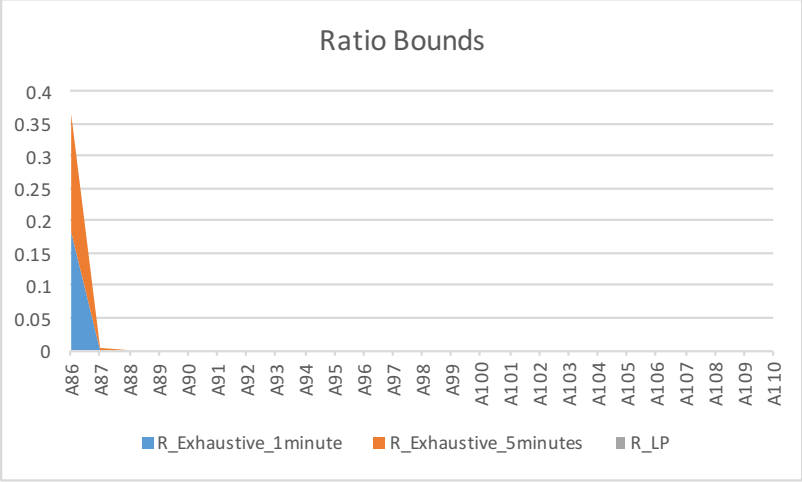

Figure 28 Ratio bounds for instance group 5

Conclusion

With the detailed results of every instances, let's consider the overall performance of our algorithms. Below table shows how often our exhaustive algorithm, ILP formula and greedy algorithm finds optimal solution.

	Overall	Instance Group 1	Instance Group 2	Instance Group 3	Instance Group 4	Instance Group 5
Exhaustive solutions are optimal (1 minute)	70.91%	100%	92%	80%	8%	92%
Exhaustive solutions are optimal (5 minute)	74.55%	100%	92%	80%	24%	92%
ILP solutions are optimal	87.27%	100%	92%	80%	80%	92%
Greedy solutions are optimal	19.09%	100%	20%	0%	0%	24%

Table 14 How often are ILP and Greedy optimal

ILP formulas could find optimal solutions for most instances while greedy algorithm couldn't find most optimal solutions. For exhaustive algorithm, the performance is very good for instance group 1, 2, 3 and 5. However, it doesn't perform well for instance group 4. Though exhaustive algorithm couldn't find optimal solution for instance 4, the solution it returned are always very near to the optimal. The reason is that the number in the group 4 instance is very large, and it need more time to calculate before finding optimal. Given that the time is limited, the exhaustive could only return the best value till the time limit.

The following graphs shows the overall ratio bounds among the exact solution, the greedy solution and exhaustive.

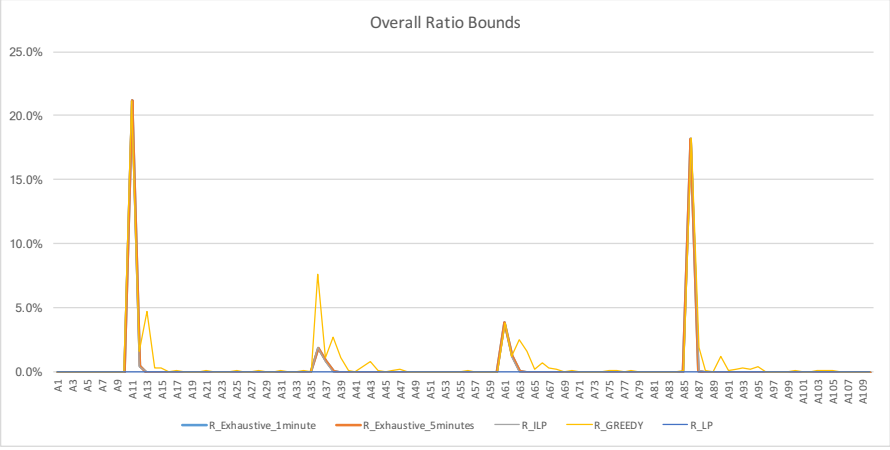

Figure 29 Overall Ratio Bounds

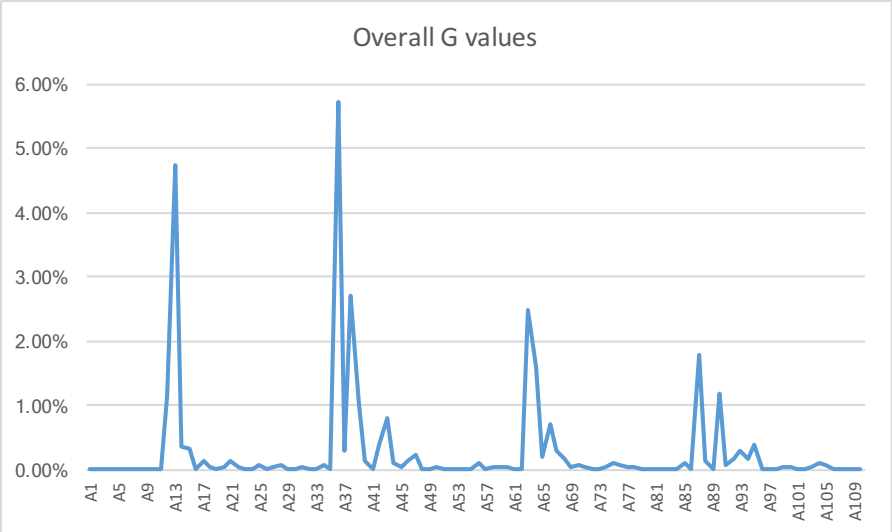

Figure 30 Overall G Values

Chapter 5

Introduction

This project will evaluate the usefulness of two different optimization algorithms. The first algorithm is steepest descent local search algorithm. We have developed two algorithms for steepest descent that find good initial solutions without speeding too much time. They are greedy algorithm and random algorithm. This project will define neighborhood of each solution based on bit sequence. The other algorithm we developed is simulated annealing.

Initialization and Neighborhood Function

Initialization

We choose two techniques for finding the initial solution: random and greedy. For random initialization, we uniformly generate a random number from 0 to 2^n-1, where n is the size of our instance. Then we convert this number to a binary array that corresponds to a subset of the instance. If the sum of elements in the subset is less than target, we know it is a valid solution. And we take it as the initial solution. For greedy initialization, we use the greedy algorithm we implemented earlier to generate the initial solution.

Neighborhood function

We generate neighborhood of a solution by changing one bit in the binary array which corresponding to a subset of the instance. For example, if S = {2, 3, 4}, target = 7, and the initial solution is {3}. Then the binary array of this initial solution will be [false, true, false], we could generate its neighbor by changing one value in the array, such as [true, true, false], [false, true, true] and [false, false, false].

Steepest Descent

Steepest Descent algorithm is a famous local search algorithm. It starts with an initial solution, while a better neighbor exists, it always chose the best neighbor. The pseudo-code of this algorithm used in this project is as follows:

```
Steepest Descent (      long startTime     // record start time in order to set time limit
                        set initial,    // Initial solution
                        Set S,          // an array of positive integers
                        Int target,     // target number
                        long timeLimit)     // set time limit here
1.    X = initial
2.    Cost = cost(X)
3.    if (within time limit):
4.          X' = find_best_neighbor(X)                  // find the best neighbor of X'
5.          if cost(X') == 0:                           // found optimal solution
6.                return X';
7.          Else if cost(X') > 0 and cost(X') < cost:   // If the best neighbor is better than current
8.                Return Steepest Descent (startTime, X', S, target, timeLimit)     // recursive searching
9.          Else:
10.               Return X
11.   Return X
```

Figure 31 Pseudo-code for steepest descent

Simulated Annealing

We choose simulated annealing (SA) as our second algorithm. Simulated Annealing is a probabilistic for approximating the global optimum. With SA, better neighbors are always selected, worse neighbors might be selected. Given a current solution X, the probability that a random neighbor X' is selected is:

$$P(select\ X') = \begin{cases} 1, if\ c(X') \leq c(X) \\ e^{\frac{c(X)-c(X')}{T}}, if\ c(X') > c(X) \end{cases}, T\ is\ the\ target\ value$$

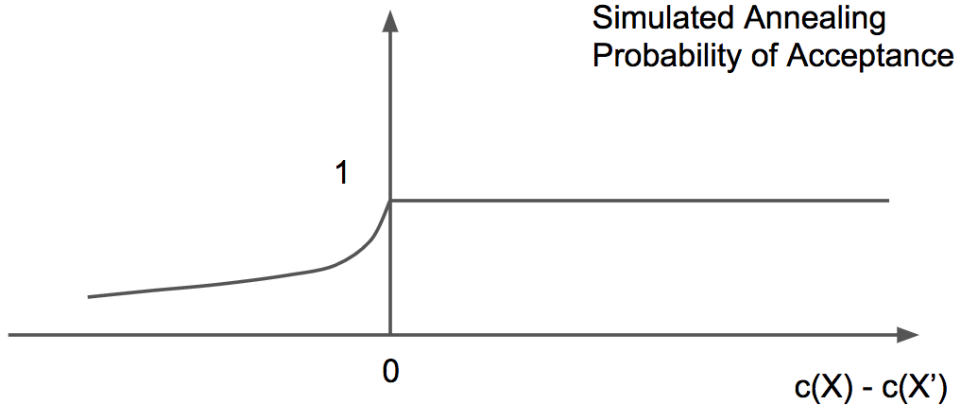

Figure 32 Probability of Acceptance

The SA algorithm used in this project is implemented in the following way:

```
Simulated Annealing(   long startTime     // record start time in order to set time limit
                       set initial,    // Initial solution
                       Set S,          // an array of positive integers
                       Int target,     // target number
                       long timeLimit)     // set time limit here
1.    X = initial
2.    Cost = cost(X)
3.    While (within time limit):
4.          X' = random_select_neighbor(X)                    // randomly select a neighbor X'
5.          If cost(X') < 0:
6.                cost(X') = T;
7.          Else if cost(X') == 0:          // found optimal solution
8.                return X';
9.          Else if cost(X') > 0 and cost(X') < cost:
10.               X = X';
11.               Cost = cost(X)
12.         Else:
13.               Num = random(0, 1)
14.               If num < e^(c(X)−c(X′)/T)

15.                     X = X';
16                      Cost = cost(X)
17    Return X
```

Figure 33 Pseudo-code for simulated annealing

This algorithm used random initialization method to produce an initial solution. The cost in the algorithm refers to the difference between the target and the current best sum found. Thus if the cost become 0 means it is the optimal solution. We have two stopping conditions: the first one is the algorithm met the time limit. Here we set time limit to be 5 minutes. The other stopping condition is the algorithm found the optimal solution.

Results, Comparison and Evaluation

In this subsection, we will only evaluate the quality of solutions between different types of Steepest Descent and Simulated Annealing. The comparison of the exhaustive algorithm and ILP, greedy and the LP lower bound will be discussed in the last session (Summary part).

Steepest Descent

For Steepest Descent, the execution time of all the instances in the benchmark didn't exceed the time limit. Therefore, we have two types of Steepest Descent: Steepest Descent with random initialization and Steepest Descent with greedy initialization.

We will list and compare the results of Steepest Descent with ILP and LP lower bound. Below graphs show the ratio bound R values of each data set.

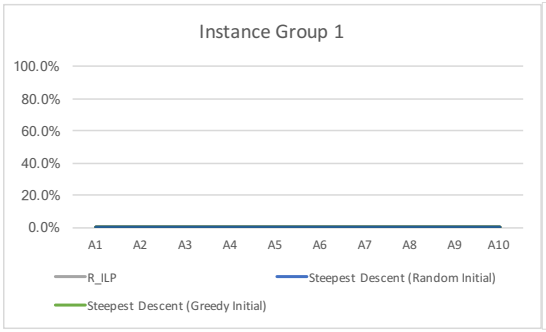

Figure 34 instance group 1

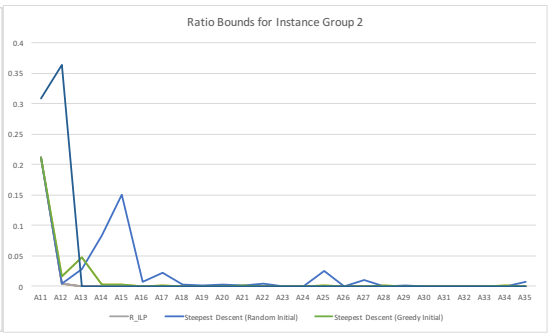

Figure 35 Ratio bounds for instance group 2

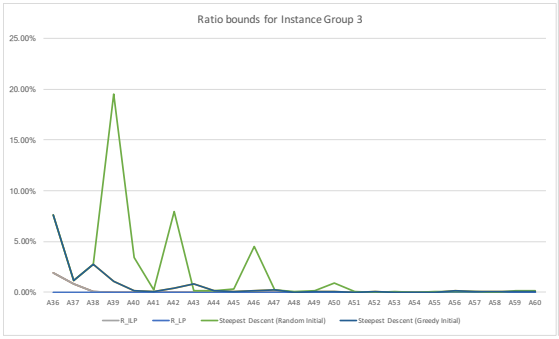

Figure 36 Ratio bounds for instance group 3

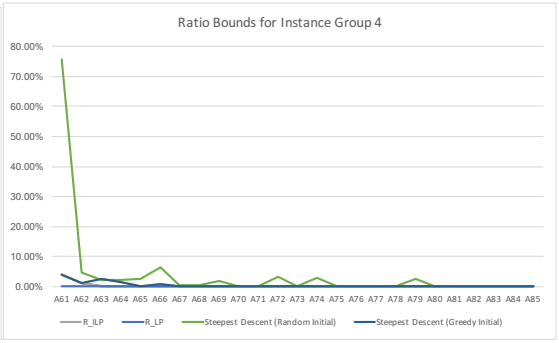

Figure 37 Ratio bounds for instance group 4

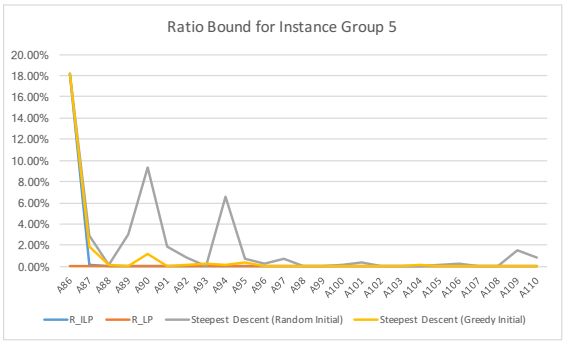

Figure 38 Ratio bounds for instance group 5

From the above result, we could see that ratio bounds for both type of Steepest Descent is tight. Generally speaking, Steepest Descent with greedy initialization has a better performance than just random initialization. With instance size gets larger and larger, Greedy Initialization Steepest Descent algorithm always has better bounds than random initialization. This is because for greedy initialization, the initial solution is already a near optimal solution, however for random initialization, the initial solution may be very far away from the optimal solution, so it is harder for this algorithm to find optimal solution.

Simulated Annealing

For simulated annealing, we chose random initialization method to find initial solution. Time limit is 5 minutes. The following is the evaluation results. In this section, we will only compare it with ILP results and LP lower bound. We will present the comparison of all algorithms in the Summary part of this report.

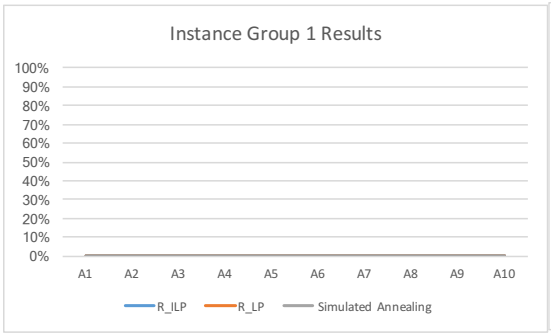

Figure 39 Instance Group 1 Results

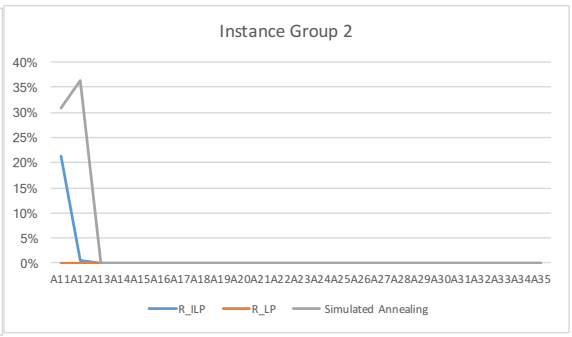

Figure 40 Instance group 2

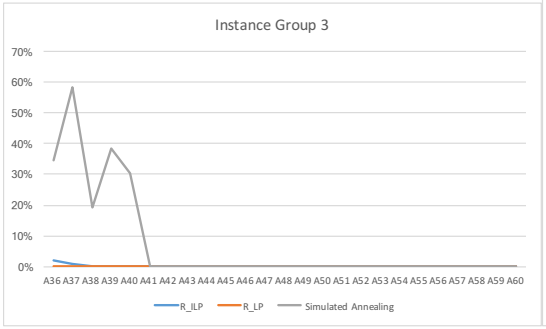

Figure 41 Instance group 3

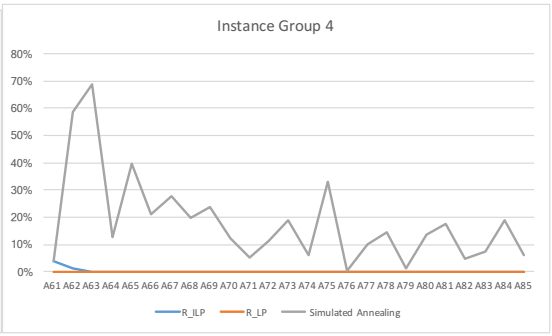

Figure 42 Instance Group 4

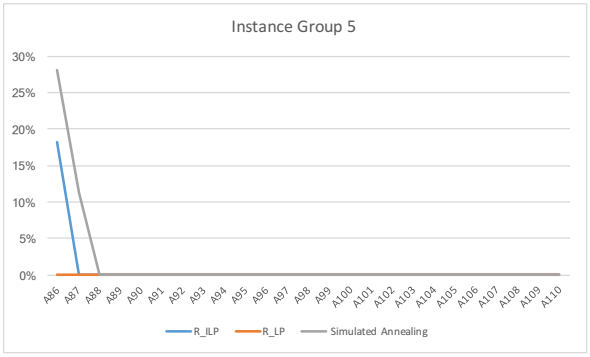

Figure 43 Instance Group 5

Just like steepest descent and greedy, Simulated Annealing could always find a near optimal solution. It performs well when instance size is small, but when instance size gets larger, like in group 4, the performance will decrease. Even though, the worse bound is just 70% from the optimal bound.

Since in simulated annealing algorithm, bad solutions may also be selected. It will find an optimal solution if no time limits. Given that our time limit is 5 minutes, it is a good choice for solving instance group 1, 2 and 5.

Summary

In this section, we will summarize our results from the entire report. Compare and contrast the results from all the algorithms we evaluated, including exhaustive algorithm, greedy algorithm, LP, ILP, Steepest Descent and Simulated Annealing.

Both the quality of solutions and runtime of algorithms will be discussed. Since our benchmark contains different types of instance, we will discuss them respectively.

Quality

Firstly, let's see how often each algorithm finds optimal solution.

	Overall	Instance Group 1	Instance Group 2	Instance Group 3	Instance Group 4	Instance Group 5
Exhaustive solutions are optimal (1 minute)	70.91%	100%	92%	80%	8%	92%
Exhaustive solutions are optimal (5 minute)	74.55%	100%	92%	80%	24%	92%
ILP solutions are optimal	87.27%	100%	92%	80%	80%	92%
Greedy solutions are optimal	19.09%	100%	20%	0%	0%	24%
Steepest Descent (Random) solutions are optimal	11.82%	100%	8%	0%	0%	4%
Steepest Descent (Greedy) are optimal	18.18%	100%	16%	0%	0%	24%
Simulated Annealing are optimal	69.09%	100%	92%	80%	0%	92%

Table 15 How often each algorithm could find optimal solution

We could see from the above table that exhaustive algorithm could give good solutions when instance size is small, but when instance size become very large, such as instance group 4, the solutions is always not optimal. ILP formula is the best on finding optimal solution. Given the time limit, greedy algorithm and steepest descent always couldn't find optimal solution. It probably because they often get stuck in local optimal solution. Simulated Annealing algorithm also has good performance when instance size is small, but when size gets larger, the performance is not satisfying enough.

Next, we could use the ratio bound R value as a criteria of the quality. The lower the ratio bound means the better the answer. Where R = 0 means that optimal solution has been found. Here we will compare the quality of the solutions of all algorithms we have researched before.

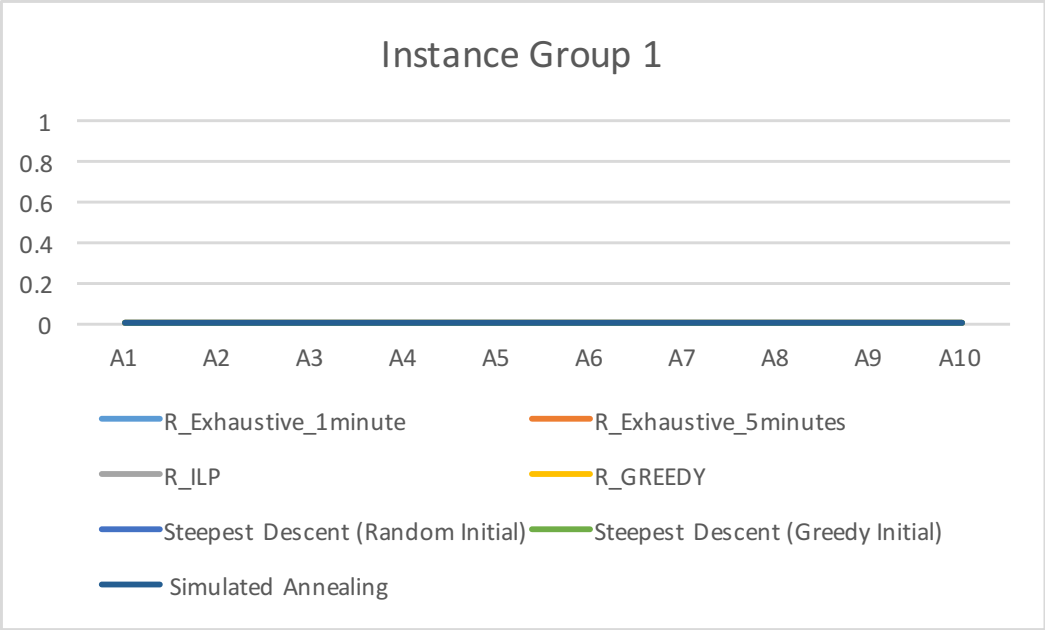

Figure 44 Instance Group 1 quality

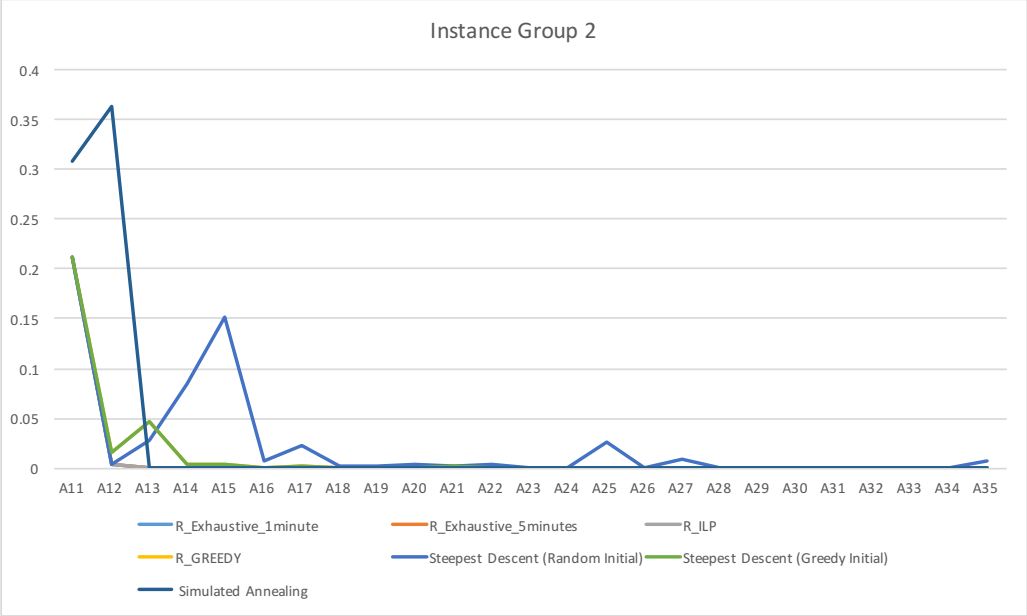

Figure 45 Instance group 2 quality

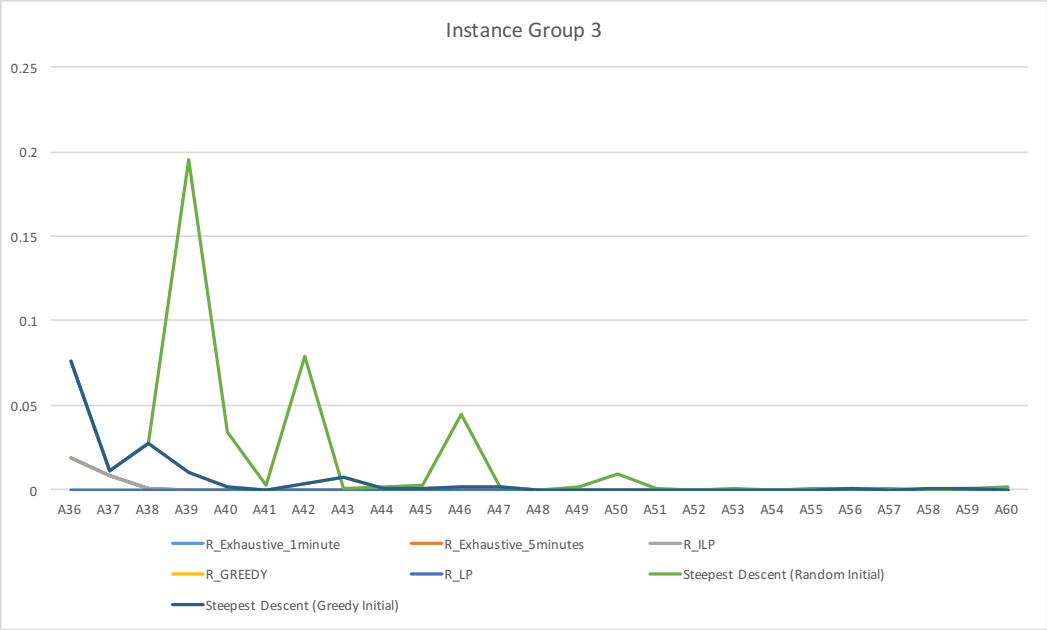

Figure 46 Instance group 3 quality

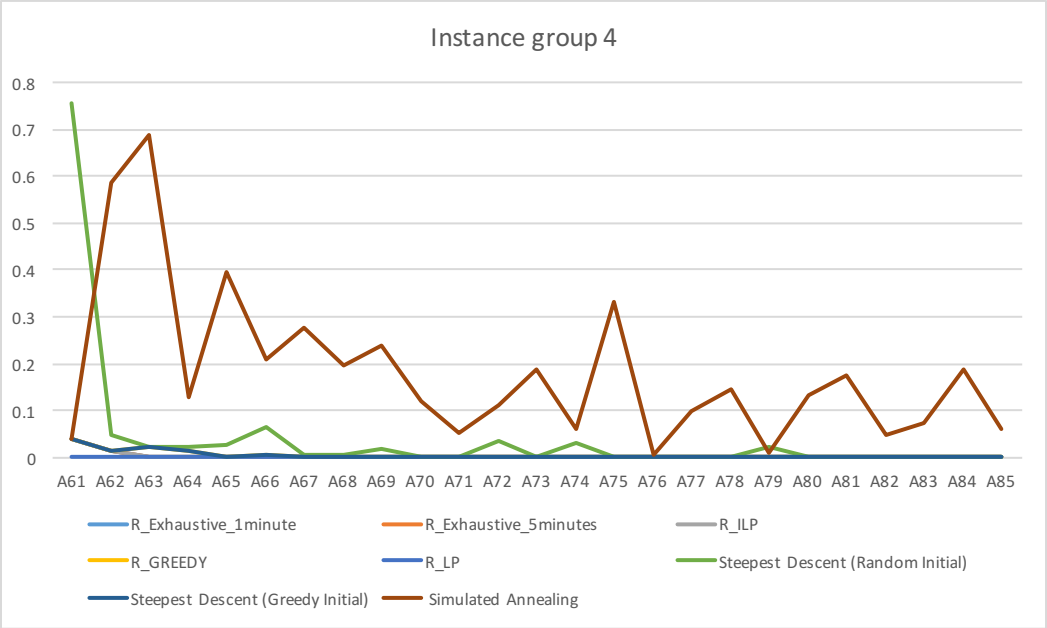

Figure 47 Instance Group 4 Quality

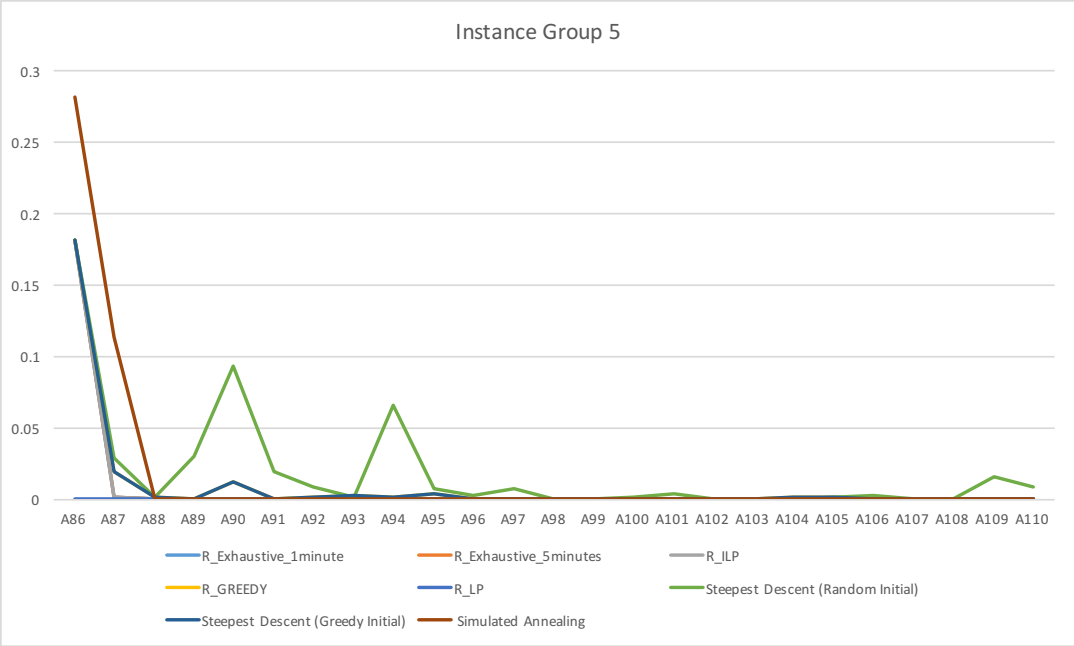

Figure 48 Instance Group 5 Quality

From the above figures, we can find that ILP formulation have a good performance on all instances. Steepest Descent with greedy initialization also performs very well. Steepest Descent with random initialization works good on instance 1, 2 and 4, works fine on group 5, but works bad on instance group 3. Probably because that since it used random initialization, some bad solution has been initialized in this group of instances. Greedy algorithm works well on finding "near optimal" answers, however, it couldn't find exact optimal answer most of time. For exhaustive algorithm, in theory it could guarantee to find an optimal answer if no time limit. In practice, within the time limit (1 minute or 5 minutes), it always could return a "near optimal" answer.

Runtimes

Here we will compare the runtimes of all the algorithms for all the instance in our benchmark. Below graphs shows the runtimes for each instance group.

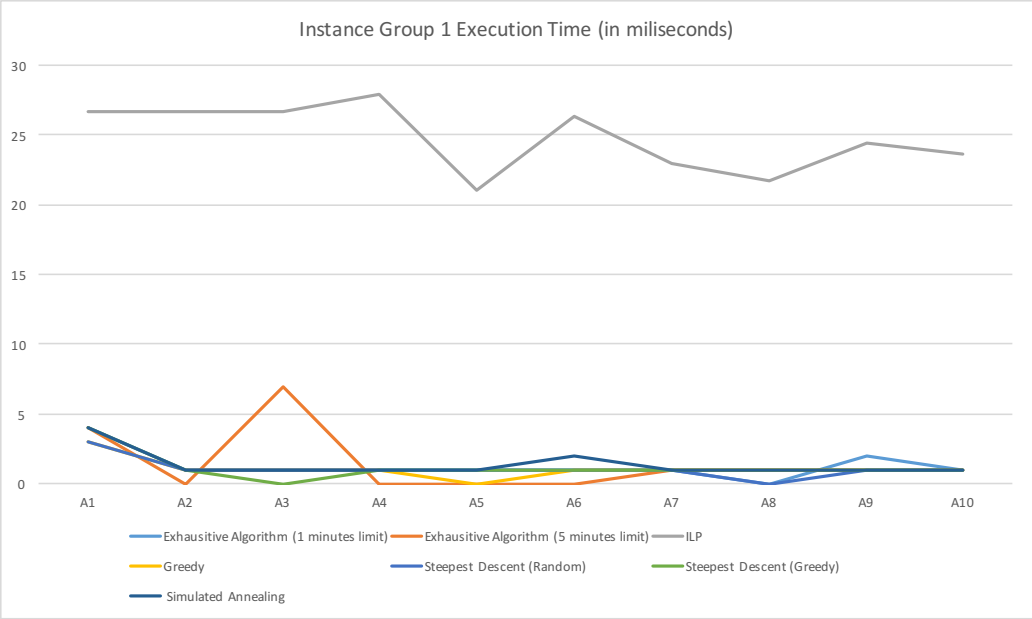

Figure 49 Group1 execution time

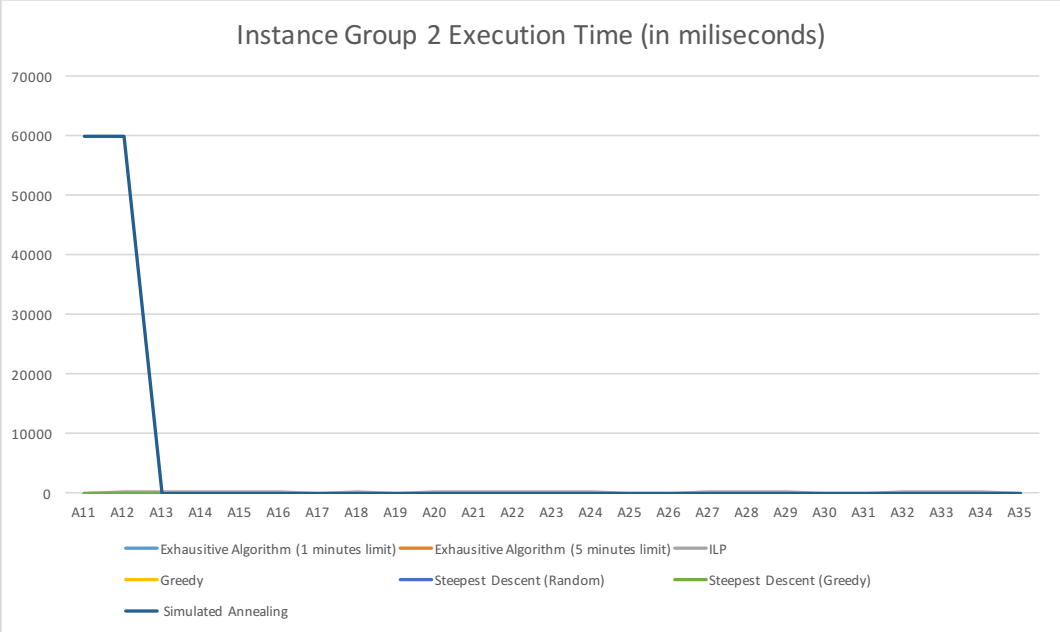

Figure 50 Group 2 execution time

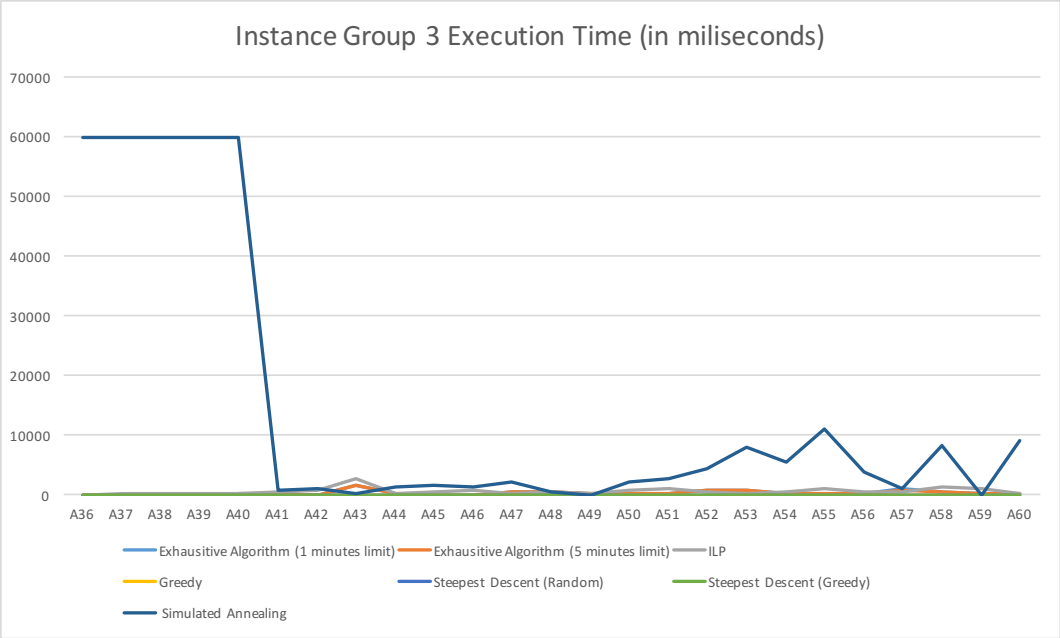

Figure 51 instance group 3 execution time

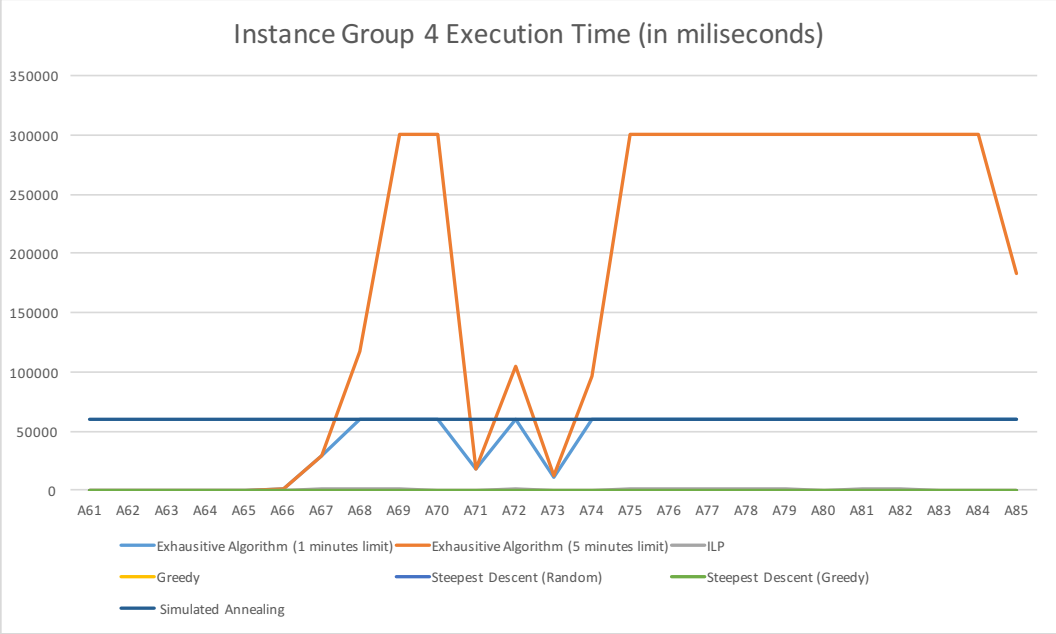

Figure 52 instance group 4 execution time

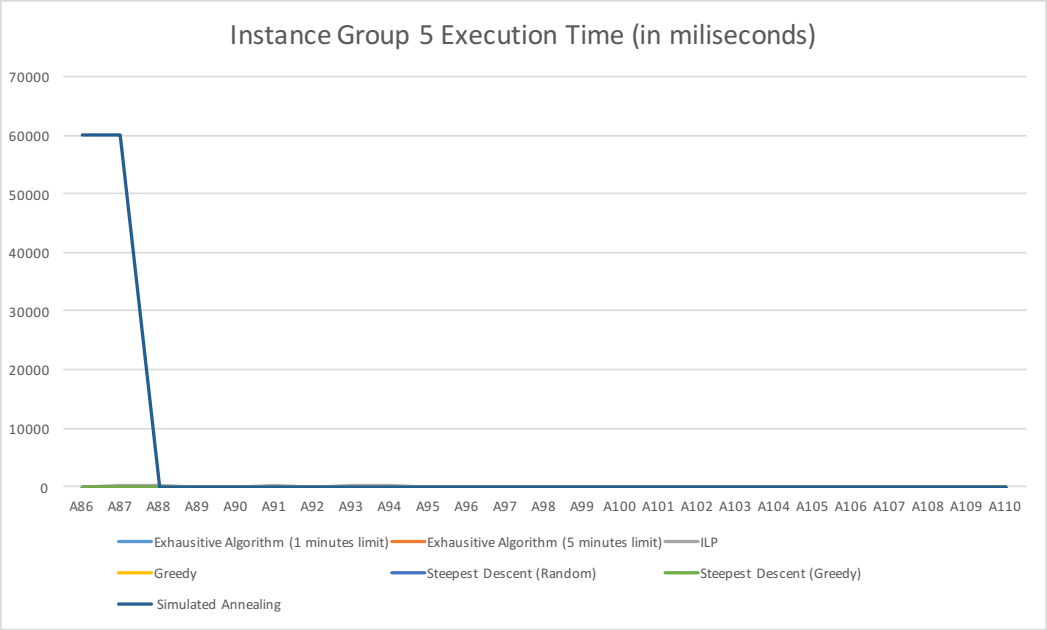

Figure 53 Instance Group 5 Execution Time

From the run times above we can know that ILP formula always runs very fast (at most in seconds). In instance group 4, steepest descent, exhaustive, simulated annealing all runs till time limit for most of instance in this group.

Next let's see separately, for each algorithm, how runtime will change with different instances:

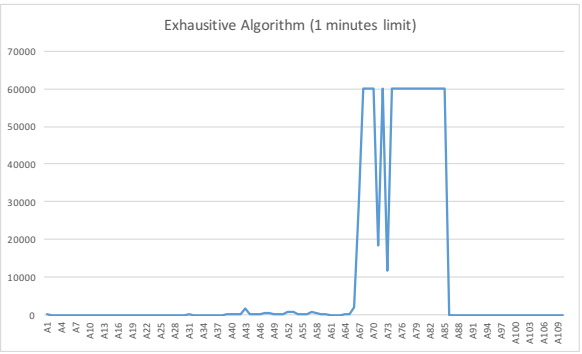

Figure 54 exhaustive algorithm (1 minute limit)

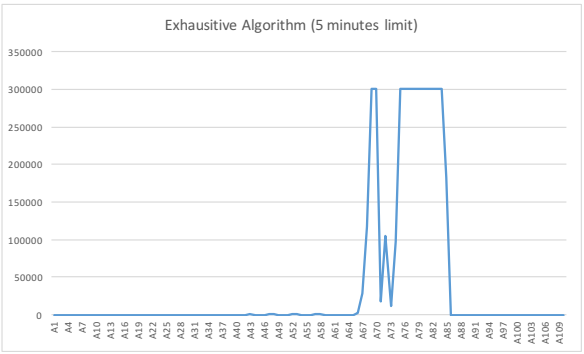

Figure 55 exhaustive algorithm (5 minute limit)

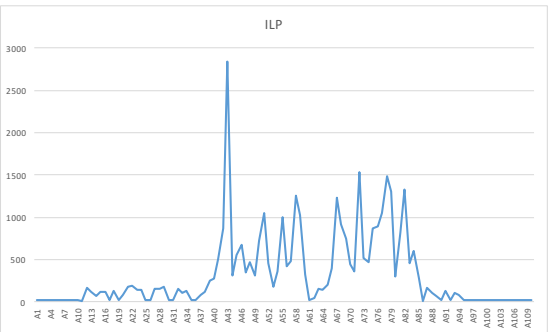

Figure 56 ILP

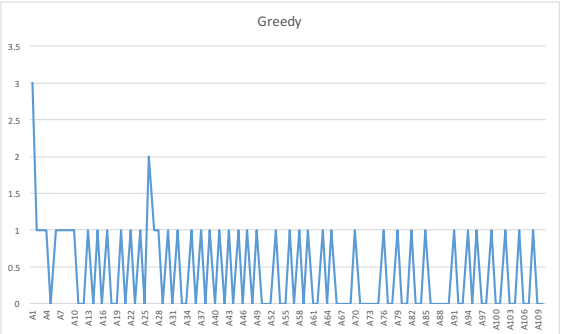

Figure 57 Greedy

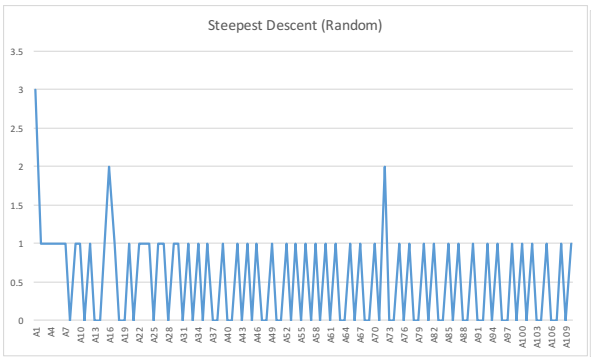

Figure 58 Steepest Descent (Random)

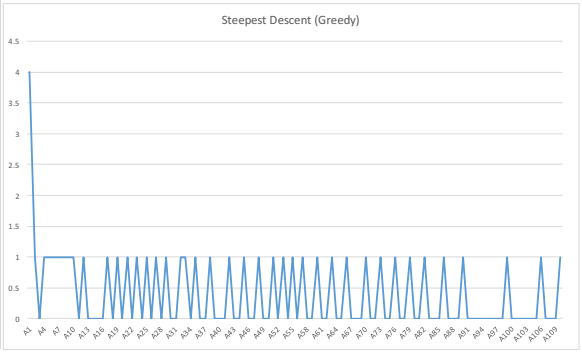

Figure 59 Steepest Descent (Greedy)

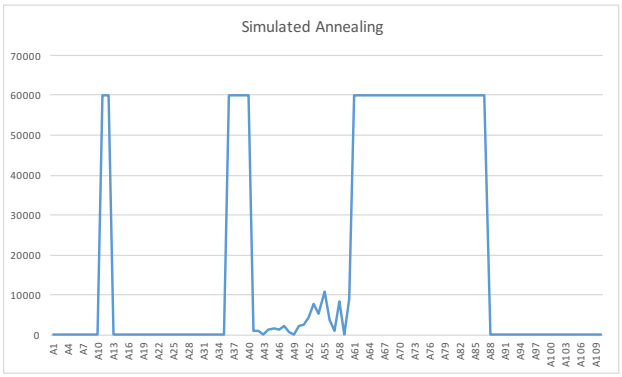

Figure 60 Simulated Annealing

We could see that exhaustive algorithm runs fast when instance size is small, but when instance size gets larger, it always reaches the time limit. ILP, Greedy and Steepest Descent runs fast on almost all instances. Simulated Annealing runs fast on small instances, but when instance size gets larger, it also will reach the time limit.

Conclusion

From the analysis above we can know that exhaustive algorithm could guarantee to find the optimal solution if without time limit, since it will brute force search every possible solution. It runs fast when instance size is small, but when instance size gets larger, it always reaches the time limit. ILP always runs very fast (at most 3 second) no matter how large instance size is, besides, it could always return optimal answer. Greedy and Steepest Descent runs fast but the solution is always “near optimal”. Simulated Annealing has a better performance on finding optimal solutions, however, when instance size gets larger, it also will reach the time limit.

In conclusion, below is our suggestion for selecting algorithms to solving instances in our benchmark:

	Instance Group 1	**Instance Group 2**	**Instance Group 3**	**Instance Group 4**	**Instance Group 5**
Recommended Algorithm(s)	All	ILP/Exhaustive/Steepest Descent (Greedy Initialization)/Simulated Annealing	ILP/Exhaustive/Simulated Annealing	ILP	ILP

Appendix

This final paper has made changes based on the comments for the previous projects given by the instructor. The main changes are:

1. Format: Include what problem we try to solve (Subset Sum Problem) in the project title.
2. In project2, this paper moved Partition problem from the "NPC sub-problems" part to "Other related problems" part, and added a new NPC sub problem called "Half Sum Problem".
3. In project4, added the exhaustive algorithm results and compared it with ILP and greedy.

Reference

[1] O'Neil, Thomas E. "0/1-Knapsack vs. Subset Sum: A Comparison using AlgoLab."
[2] NP(complexity). Wikipedia[Online]:The Free Encyclopedia. http://en.wikipedia.org/wiki/NP (complexity).
[3] Jain, E., Jain, A., & Mankad, S. H. (2014, September). A new approach to address Subset Sum problem. In Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference- (pp. 953-956). IEEE.
[4] Isa, H. A., Oqeili, S., & Bani-Ahmad, S. (2015). The Subset-Sum Problem: Revisited with an Improved Approximated Solution. International Journal of Computer Applications, 114(14).
[5] M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP Completeness, Freeman, 1997.
[6] A. Gajentaan and M. H. Overmars, On a class of O n! problems in computational geometry, Comput. Geom. Theory Appl. 5 (1995), 165-185
[7] O'Neil, T. E. An Empirical Study of Algorithms for the Subset Sum Problem. In Proceedings of the 46th Midwest Instruction and Computing Symposium (LaCrosse, WI, 2013).