

Final Report

Anh Vu Nguyen

April 26, 2016

Introduction

Throughout the semester, we have gone through different reductions and different way to solve Knapsack problem in reasonable time. This experiments aim to display the differences quantitatively in terms of algorithms' result as well as running time.

There are three parts to this experiment:

1. **A general assessment of the value returned from the four algorithms implemented to solve Maximum Knapsack problem using 100 large random instances.**
2. **Assessment on running time using large random instances of Maximum Knapsack.**
3. **Assessment on running time involving large random instances of 3SAT, which are reduced into Subset Sum, and solved using algorithms for Maximum Knapsack problem.**

Notation

This report will use abbreviations to make it easier to know what algorithm that is being discussed:

- DP1 – This refers to the $O(nW)$ dynamic programming algorithm that we discussed in CS305.
- DP2 – This refers to the $O(n^2 \cdot v(a_{max}))$ dynamic programming algorithm from the textbook based on the MinCost version of the problem.
- Greedy – This refers to the greedy 2-approximation from the textbook.
- FPTAS – This refers to the FPTAS based on scaling with the optimal dynamic programming algorithm from DP2.

Experiment Setup

The algorithms to generate running time are taken from pseudocode provided in CS305 as well as from “What is a Computer and What Can It Do?” (O’Connell). In total, I implemented four algorithms for solving Maximum Knapsack problem and also two reductions for decision problem (3SAT, 1in3SAT, and SubsetSum).

Since the reports focus on two different experiments, I set up two different workflows to accomodate this.

Maxxim Knapsack experiment

For this experiment, I set up 100 random instances of Maximum Knapsacks. There was a need to arbitrarily curb the maximum values for number of item, as well as each item’s value and cost in order to make sure that each instances runs in reasonable amount of time (below 5 seconds for each algorithm). Each knapsack problem is constrained to the following attribute:

- At most 200 items

- Each item's value cannot exceeds 1000 and is an integer
- Each item's cost cannot exceeds 1000 and is an integer

Using this constraints, I also limit the range of Maximum Knapsack problem presented so that we have a sample with specified attributes. This limits the problem we would face when we have to group instances with 50 items and 5000 items should we not enforce the constraint. The calculation of density of instances then would not be as reflective of the actual algorithms due to skewness introduced by instances with a very large number of items.

Each instances solved using the four algorithms for solving Maximum Knapsack. The results (maximum value returned) as well as running time from each algorithm is logged into a text file (can be found in log folder of this project). I also wrote a simple Python script (can be found in scripts folder of this project) to parse this log file into a csv file for statistical analysis with R. This report as well as all the graphs are generated using R.

Reductions For 3SAT

For this experiment, I set up 100 instances of 3SAT. Each instance is then reduced to 1in3SAT, which is then subsequently reduced into Subset Sum and Knapsack respectively. The running time needed to solve the Knapsack instances and returns an answer for the decision version of the problem is recorded (the time for the reduction is not taken into account as this would take same amount of time for all algorithms). All timing is still recorded in ms in order to maintain consistency.

Since the program is written in Java, where number is constrained to 64-bits, there is a limit to how many clauses and variables I can have in 3SAT instances. This is due to the way we performed our reduction. When reduced from 3SAT to 1in3SAT, we know that the number of clauses will triple, and for each clause will introduce four new literals as well. When we reduce 1in3SAT, the total number of literals and clauses makes up the target for Subset Sum. Therefore, I have instances where we either exceed Java Heap Space while doing the reduction, or we ran out of number (and loop back to negative number, which breaks the algorithm). Therefore, there's first a need to perform a check on the maximum number of clauses 3SAT instances can have. Since the assignment of literals in clauses is randomized in my algorithms, I do not have control on this for now. After testing with several instances, I decide to limit the number of clauses to 2 clauses.

I have also tried to come up with way to reduce the size of number that are passed in for Subset Sum instances. In the original reduction, we use the number of literal for v and v' in the digit that matches the literal number (e.g. literal #4 will have 1 in the 4th digit of the variable v and v' generated from this literal). The reason for this is to make sure that we do not use the same item more than once, which would mean that we chose true and false for the same literal. In order to reduce the size, I took out all the column representing the literal number. Instead, since we are adding v and v' into our list respectively, we use index in the array to check for potentially taking two items that we cannot take. After using Knapsack to solve for this problem, we would go through the set and check to see whether there're any two consecutive items both present in the list of item taken. If there's, we reject the results from Knapsack. This way, we would only have to deal with the number of clauses as values for v and v' . However, this only helps us to get to 2 clauses in the original 3SAT instances.

At the same time, in order to compensate for the small number of clauses in 3SAT, I generated 200 random instances instead of 100 like the Knapsack experiments. The larger sample size hopefully can give us a better understanding into the running time for this experiment.

The Four Algorithms for Maximum Knapsack

The expectation for the four algorithm being implemented to solve Maximum Knapsack problem are summarized as follow:

1. DP1 is expected to run slowly compared to the other algorithms.
2. Greedy is expected to run fastest compared to all other algorithms.
3. Since FPTAS is implemented based on DP2, the expectation is that the running times of the two are very similar.
4. As FPTAS scales the values of item, the expectation is that it would run faster than DP2 due to smaller size of table built along the way.

Assessment of Maximum Knapsack Algorithms

Maximum Knapsack Returned Values

Values Returned from DP1

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	72	5066	10390	19350	31690	81360

Values Returned from DP2

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	0	4983	10250	19290	31480	81340

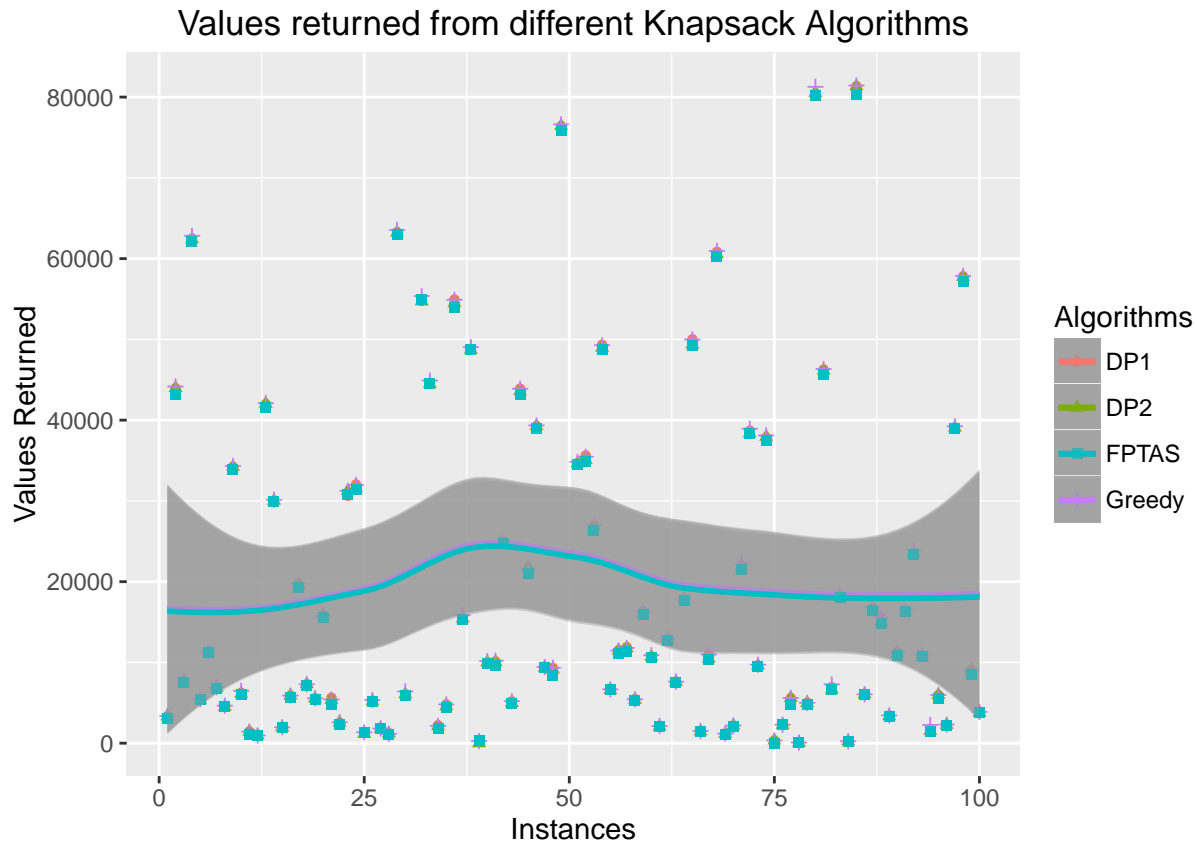
Values Returned from Greedy

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	83	5055	10560	19450	31800	81430

Values Returned from FPTAS

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	0	4758	10100	19070	31300	80330

Below is the graph plotting the returned values for each instances. The color indicates which algorithms is being used. At the same time, there's a trend line indicating the trend of values for each algorithm.



From the graph, we hardly see any differences in the values returned from the four different algorithms. The greedy algorithm seems to have the highest values returned, while DP2 returned the lowest. We cannot see the trend lines for DP1 as it is likely to be overlapped by another line.

Running Time

The following captures the general statistics of the running time of different algorithms (everything is in ms).

DP1 Running Time

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	0.062	1.850	7.537	45.100	54.410	395.500

DP2 Running Time

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	0.054	33.920	114.000	360.800	615.700	1994.000

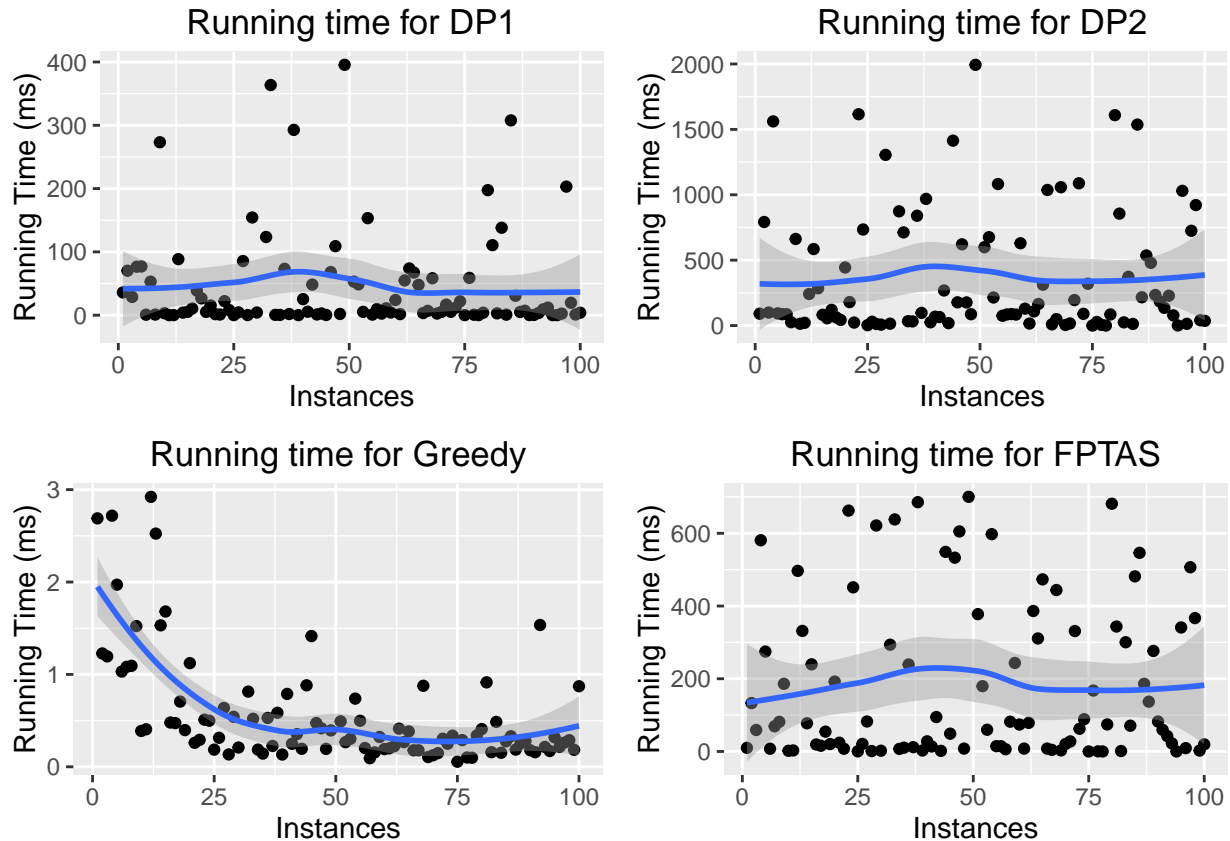
Greedy Running Time

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	0.0550	0.1962	0.3270	0.5634	0.6240	2.9220

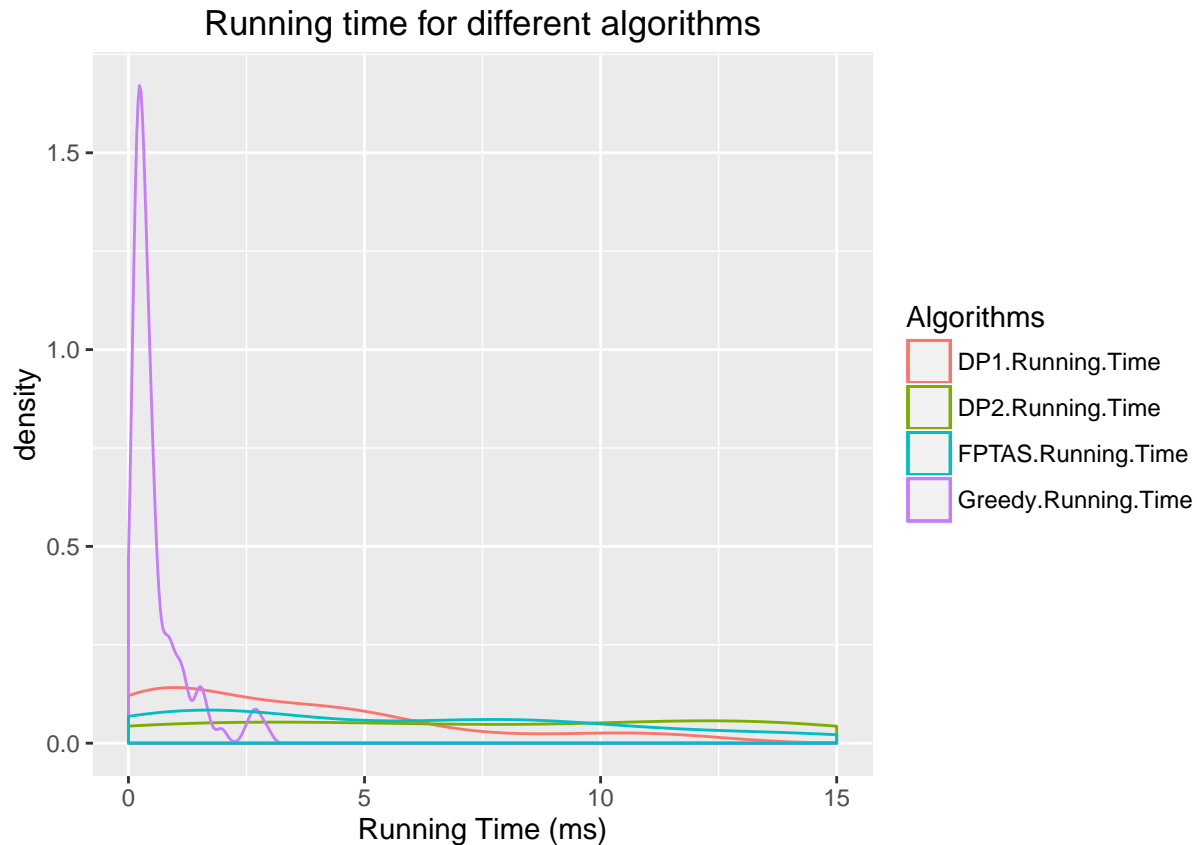
FPTAS Running Time

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	0.046	10.830	74.260	181.800	326.200	700.300

Below is the graphs of running time for different instances. There is a smooth line to indicate the general trend of all the instances. Noticing the difference on the y-axis, which indicates the running time in millisecond (ms), we can see that Greedy runs in the range below 3ms while DP2 runs in the range 2000ms, or 2 seconds. FPTAS seems to have a higher upper bound for running time compared to DP1. FPTAS and DP2 graphs are very similar, despite the differences in scale on the y-axis. This is somewhat expected as FPTAS implements DP2 as part of its algorithm. From this graph alone, we would expect that Greedy would have the best running time.



Below is the density graphs for the running time of the four algorithms with our 100 instances. This graphs gives us information on how the running time is distributed among the testing instances.



From the graph, we can see clearly that Greedy stands out as having the most of its running time in the lower spectrum of the graphs. This aligns with our expectation that this would have the best running time. DP1 running time at the same time does not span as much as the DP2 and FPTAS.

Reduction For 3SAT

This analysis is focused on running time for the following sequence of actions:

1. Reduce 3SAT into 1in3SAT.
2. Reduce 1in3SAT into Subset Sum.
3. Reduce Subset Sum into 0-1Knapsack
4. Solve 0-1Knapsack using values returned from the four algorithms with Maximum Knapsack.

Running Time

Below are the statistical summary for the running time of each algorithm (everything is in ms).

DP1 Running Time

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	65.02	85.48	96.31	113.40	126.50	386.60

DP2 Running Time

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	3344	4863	5646	6108	6775	16940

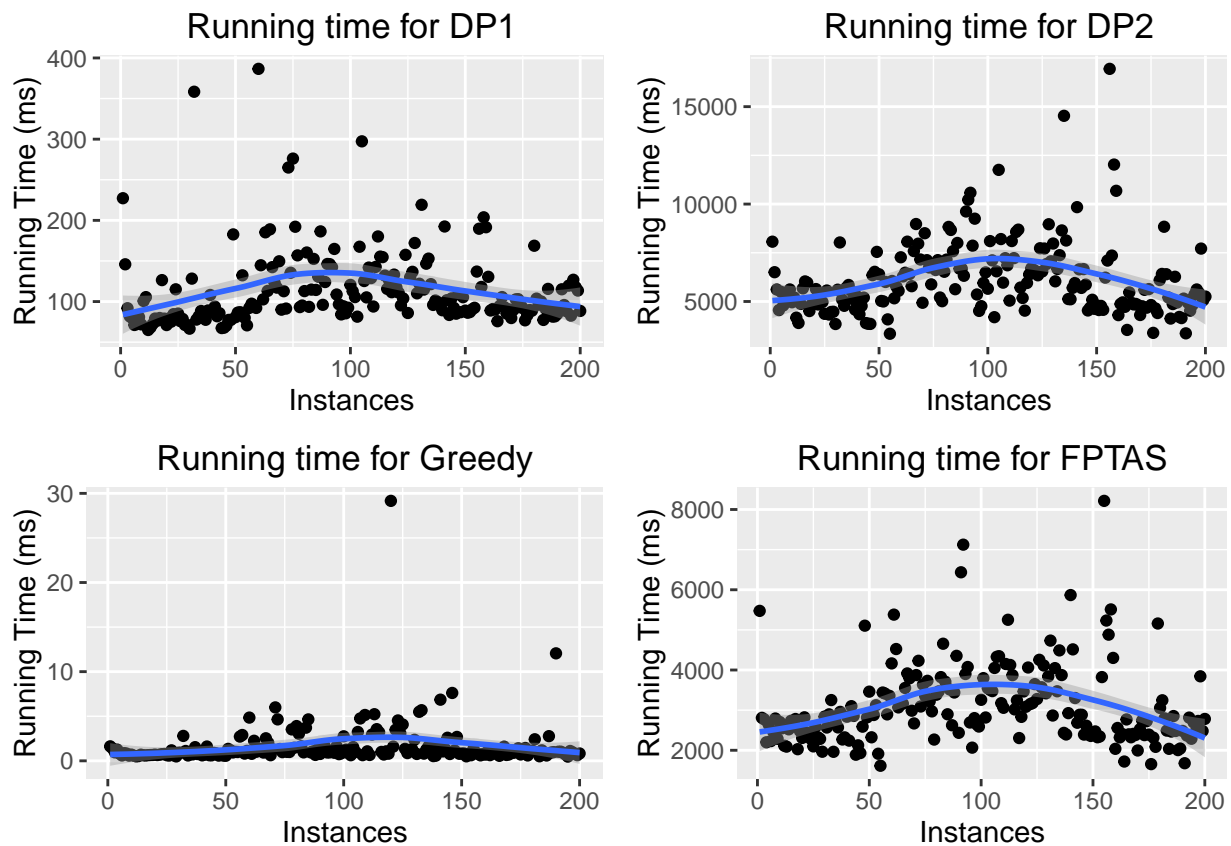
Greedy Running Time

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	0.4720	0.7103	0.9680	1.6300	1.4180	29.1600

FPTAS Running Time

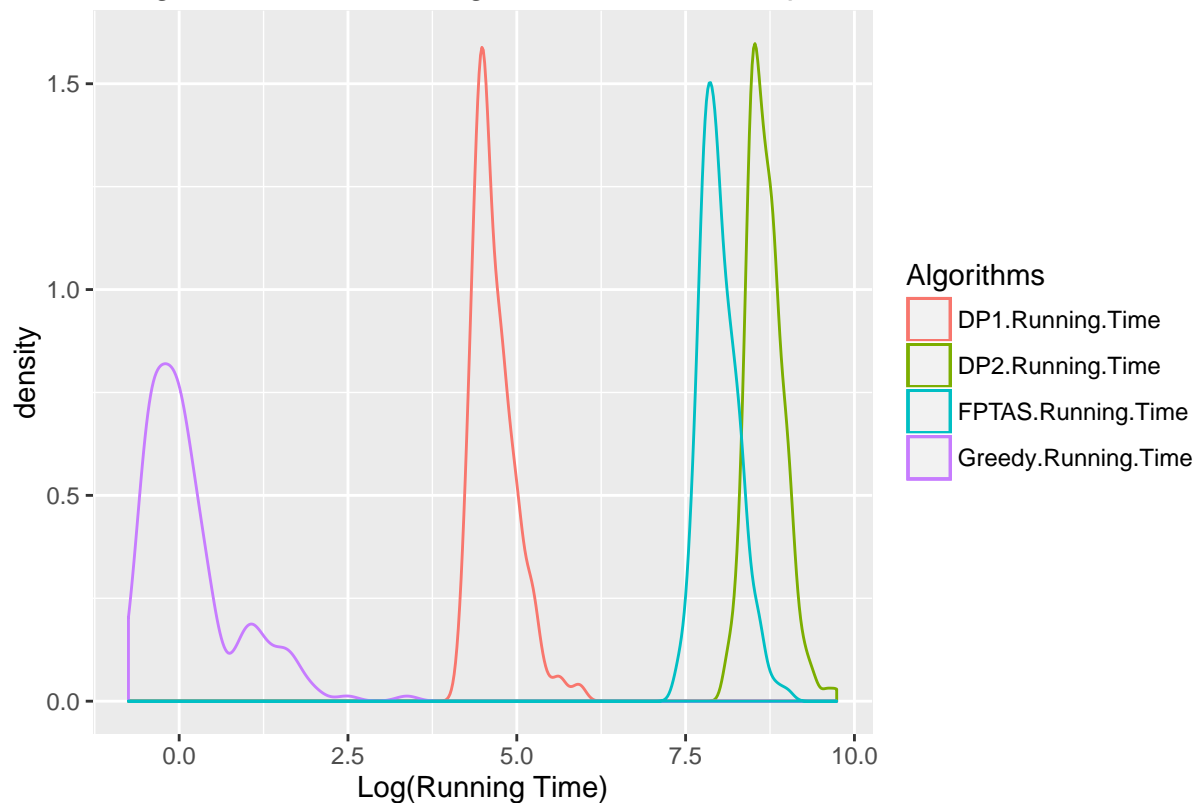
##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	1614	2396	2820	3093	3507	8215

Below are graphs of the running time of the instances with the trend lines in each graph.



Below is the density graphs for the running time of the four algorithms with our 200 instances. This graphs gives us information on how the running time is distributed among the testing instances. Since there is such a disparity between the distribution of running time between Greedy and DP2 and FPTAS, I decided to take the log scale of the running time of each instances.

Running time for different algorithms for 3SAT experiment



Improvements For Experiments

There are definitely rooms for improvements with this experiments that I could have done in order to yield better comparison.

Firstly, I could have increased the sample size from 100 to 1000, which should give us a better understanding regarding the values and running time. However, due to the circumstances of limited time frame to conduct this experiment, the limit of 100 instances in each experiment should be sufficient to give us a survey into our investigations.

Secondly, we could have implemented the same algorithms in different languages such as Python and C. By taking the average of the same algorithm in three different languages, it would help us get a running time that is pseudo language-agnostic. The average for this number can be compared in the same manner in order to get the differences between different algorithms.

Conclusion

References

1. O'Connell, Thomas C. "What Is a Computer and What Can It Do?" Place of Publication London: College Publications, 2013. N. pag. Print.