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 100 large random instances of Maximum Knapsack.

3. Assessment on running time involving 100 large random instances of 3SAT, which are reduced into Subset Sum, and solved using alogirthms 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-approximatino 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.

Maxmim 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 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 Four Algorithms for Maximum Knapsack

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

1.
2.
3.
4.

Assessment of Maximum Knapsack Algorithms

Maximum Values Returned

Running Time

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

DP1 Running Time

```
##       median         mean      SE.mean CI.mean.0.95          var
##     7.537000    45.098051     8.038876    15.954944  6333.105878
##      std.dev     coef.var
##    79.580814     1.764618
```

DP2 Running Time

```
##       median         mean      SE.mean CI.mean.0.95          var
## 1.140240e+02 3.607797e+02 4.695299e+01 9.318868e+01 2.160491e+05
##      std.dev     coef.var
## 4.648108e+02 1.288351e+00
```

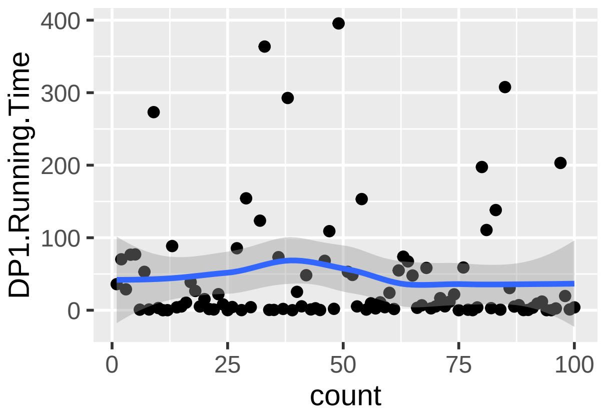
Greedy Running Time

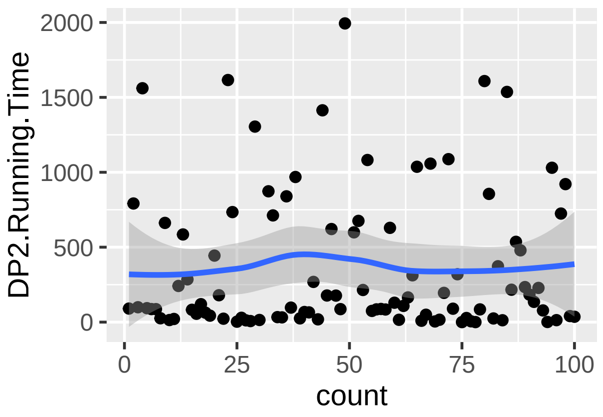
```
##       median         mean      SE.mean CI.mean.0.95          var
##   0.32700000   0.56343878   0.06048931   0.12005453   0.35857771
##      std.dev     coef.var
##   0.59881359   1.06278377
```

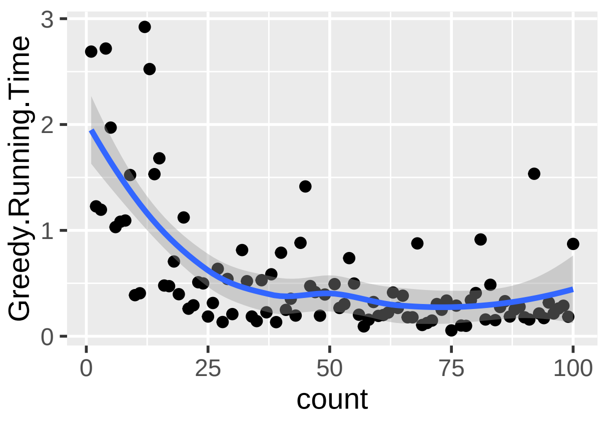
FPTAS Running Time

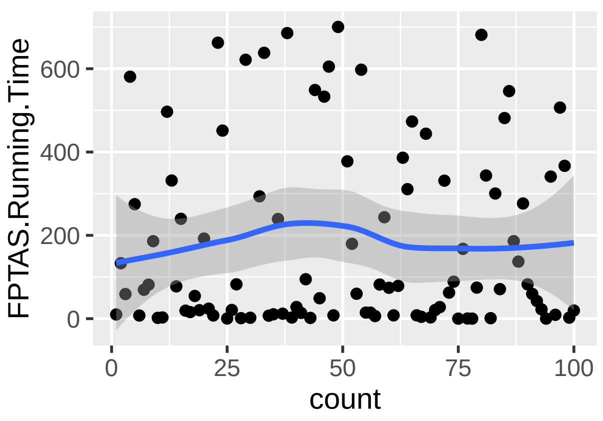
```
##       median         mean      SE.mean CI.mean.0.95          var
##    74.265000   181.798306    21.783232    43.233687 46501.903368
##      std.dev     coef.var
##   215.643000     1.186166
```

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.

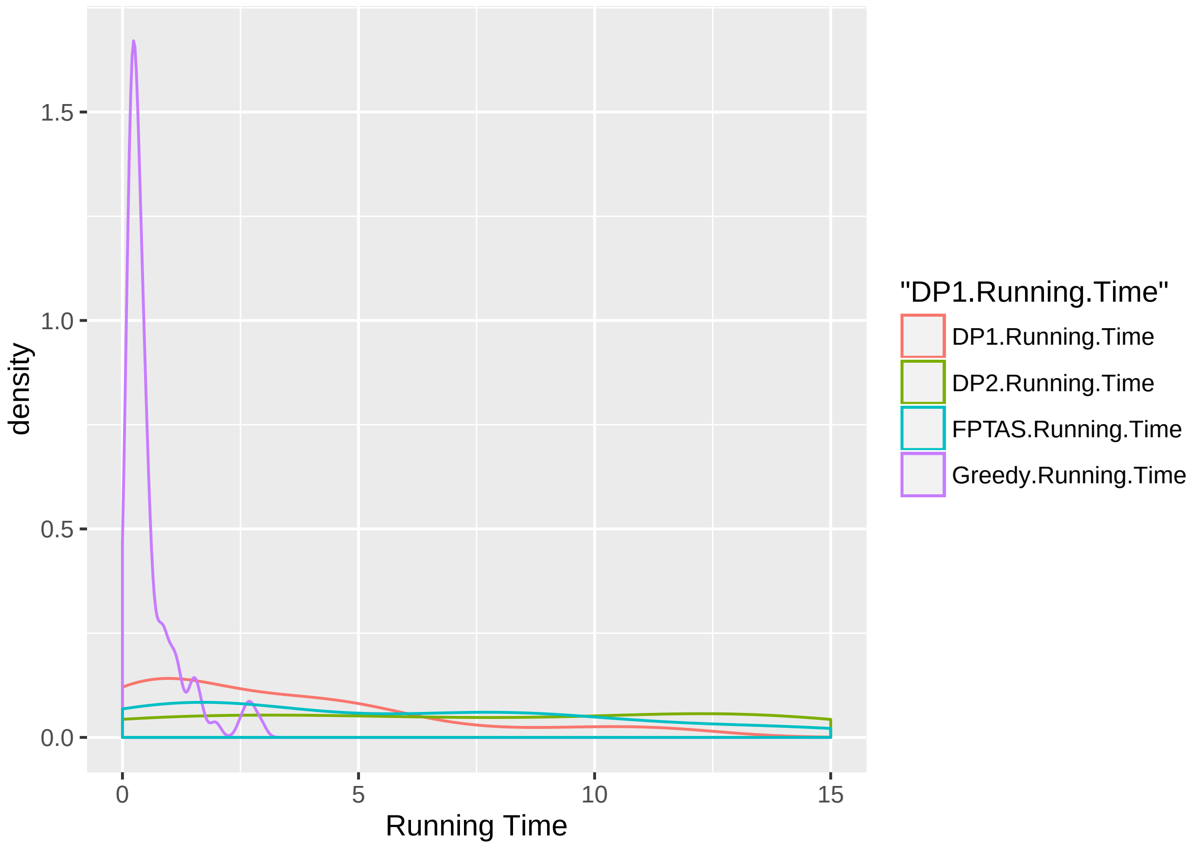








Below is the density graphs for the running time of the four algorithms with our 100 instances. This graphs gives us information on how the running time is distributed among the testing instances.



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