Troy Daniels

Computational Complexity Experiment

Introduction:

Computers capabilities in solving problems today are often limited by the complexity of the problem. As a result we categorize problems by the asymptotic time they take to solve. We consider problems with exponential time complexity to be in NP. These problems have proven to be very hard to solve. (for large instances) This experiment is testing how well many approximation algorithms help us solve NP-hard problems. More specifically, the Maximum 0-1 Knapsack, Satisfiability and Traveling Salesman optimization problems. For each one, the runtime and optimality of the approximation algorithms is recorded on many different randomized instances of the problem varying in size and sometimes other variables. The average, best, worst and median runtimes and optimality are analyzed. As well as the asymptotic growth of time for each approximation algorithm as the instance size grows. The program created to run this project is in Java. In order to run the full program, the code editor Eclipse must be used because of dependencies. The files that run the tests are "RunningSat.java", "RunningTSP.java", "RunningKnapsack.java" and "KnapsackLargeTest.java". Each test outputs a text file with all the data recorded. All data (text files) for the tests below are attached.

Maximum 0-1 Knapsack:

Algorithms:

These tests record the runtime and optimality of the Zero One dynamic programing knapsack- O(nW), Min Cost dynamic programming knapsack - O(n^2(V(a_{max}))), 2-approximation maximum knapsack - O(nlgn) and two FPTAS maximum knapsack with different epsilons (0.4,0.8) - .O(n^3*(1/**ε**)).

Notes:

- One set of tests (Test #1) start at an instance size of 50 and go up to 350 (inclusive). For each instance size each algorithm solves 100 random instances and then averages out the optimality and runtime.
- The other set of tests (Test #2) is on an instance size of 300. It solves 100 randomly generated instances and computes the best, worst, average and median for both optimality and runtime.
- Optimality is a number between 0 and 1. For each approximation algorithm on each instance, its solution is divided by Zero One's solution.
- Each instance of knapsack is randomly generated but the max values for weight and value are bounded between 1 and n*3. The capacity is also bounded by 1 and n*4.
- For the 2-approximation maximum knapsack, Quicksort is used to order the values/weights in descending order.
- For all instances solved in the test, min cost and zero one had the same solution.

Test #1:

Runtime (average)

Instance Size	Zero One	Min Cost	2-Approx	FPTAS (0.4)	FPTAS (0.8)
50	0.98	1.86	0.12	1.61	1.06
100	6.18	16.92	0.01	12.6	7.19
150	17.48	39.16	0.03	32.27	16.31
200	49.05	82.79	0.09	70.38	34.0
250	216.23	443.08	0.21	371.73	193.33
300	536.84	1752.95	0.71	1341.36	721.28

Optimality

	Zero One	Min Cost	2-Approx	FPTAS (0.4)	FPTAS (0.8)
50	1	1	0.963741	0.999924	0.999815
100	1	1	0.966877	0.999987	0.999964
150	1	1	0.970279	0.999992	0.999981
200	1	1	0.967768	0.999995	0.999987
250	1	1	0.974551	0.999996	0.999992
300	1	1	0.9719	0.999998	0.999996

Test #2:

Runtime (instance size of 300)

	Zero One	Min Cost	2-Approx	FPTAS (0.4)	FPTAS (0.8)
Average	446.43	1260.93	0.61	1851.41	746.27
Best	26.0	406.0	0.0	331.0	164.0
Worst	1364.0	3417.0	6.0	4291.0	3447.0
Median	364.0	1163.0	0.0	2009.0	355.0

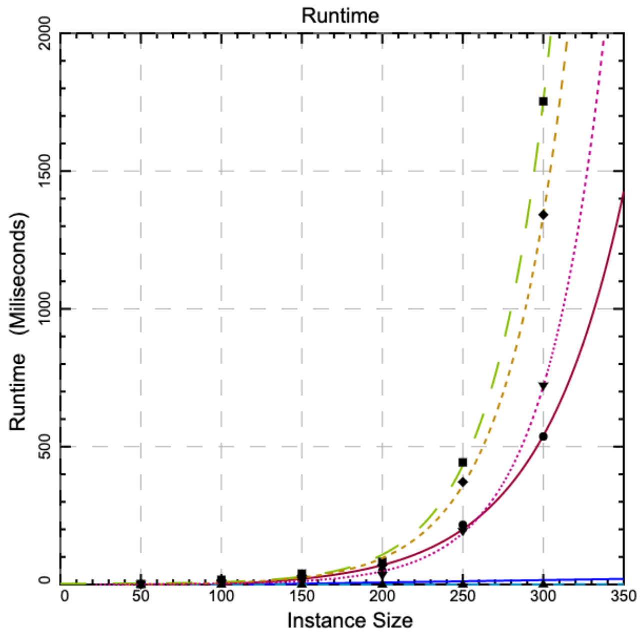
Optimality (instance size of 300)

	Zero One	Min Cost	2-Approx	FPTAS (0.4)	FPTAS (0.8)
Average	1	1	0.965362	0.999998	0.999996
Best	1	1	1	1	1
Worst	1	1	0.878665	0.999989	0.99965
Median	1	1	0.979340	1	1

Analysis:

The runtime results for test #1 is as expected. Zero One dynamic programming, Min Cost and FPTAS runtimes all grow at a very fast rate. It is clear that as the instance size keeps growing the time to run one of these algorithms will be unreasonable. For the 2-approximation, however, we see a slow growth in time. And so, is capable of solving very large instances in short amounts of time. Examine the graph below.

Blue line = 2-approximation, Solid red line = Zero One, Dotted red line = FPTAS (0.8)
, Dashed red line = FPTAS (0.4), Dashed Green line = Min Cost



Now, looking at the optimality of these algorithms in test #1, we see that the approximation algorithms get very close to the optimal answer. For the FPTAS with a large epsilon, we see that it greatly improves the running time (over Min Cost) but only minimally affects the optimality. For this runtime and optimality tradeoff, we see that the 2-approximation is clearly our winner. It is able to usually get a very close optimal answer (average is >0.95) , extremely fast. If you are in a situation where you need to solve the knapsack problem and can afford some error in optimality over many instances, the 2-approximation is the algorithm to use. However, Test #2 shows us that for individual instances the 2-approximation can give a greater error. Over the 100 instances ran even though the median had very low error (0.979), on one instance the error was much higher (0.878). Whereas for the FPTAS with high epsilon, it shows low error for all instances it ran on (worst is 0.99). It also was able to find the optimal answer most of the time (median of 1). Also notice that the FPTAS has a lower running time over Zero One knapsack while the instance size is low. The question then, is can you raise the epsilon high enough such that it runs faster than Zero One knapsack on a specific instance size while keeping low error for all instances. Otherwise, Zero One knapsack is the best choice.

SAT:

Algorithms:

These test record the runtime and satisfiability of DPLL, GSAT, ⅞ randomized approximation and randomized rounding linear programming.

Notes:

- 5 different tests are run on different ratios of the number of clauses to the number of variables. The ratios are ½, 1, 2, 4.26 and 6.
- For each ratio the instance size starts at 20 variables and increases by 20 until 80 variables.
- For each instance size 100 randomly generated instances are solved by each algorithm and the runtime/satisfiability average,best,worst and median are calculated.
- For each instance the algorithm satisfiability score is the number of clauses satisfied divided by number of clauses.
- All instances are completely randomized 3-Sat instances with fixed number of clauses and variables.
- DPLL satisfiability only keeps track of instances that are satisfiable.
- GSAT makes only 10 changes to variables.
- I only created tables for info that I found prevalent (lots of data here).

Runtime (average, clause ratio of 1.0)

# of variables	DPLL	GSAT	Random Rounding	Seven Eighths Approximation
100	0.47	0.11	31.63	0.04
200	0.88	0.16	158.49	0.08
300	2.16	0.38	788.01	0.32
400	4.91	0.75	1920.49	0.69

Runtime (average, clause ratio of 4.26)

# of variables	DPLL	GSAT	Random Rounding	Seven Eighths Approximation
20	0.2	0.02	15.39	0.01
40	1.98	0.09	129.73	0.02
60	18.29	0.08	486.11	0.03
80	159.47	0.25	1320.17	0.09

Runtime (average, clause ratio of 6.0)

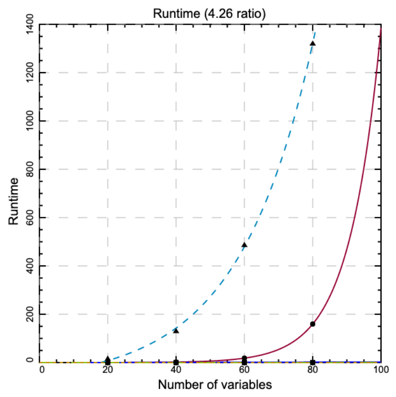
# of variables	DPLL	GSAT	Random Rounding	Seven Eighths Approximation
20	0.22	0.06	47.35	0.00
40	2.16	0.05	416.44	0.00
60	11.24	0.22	1447.36	0.08
80	46.84	0.36	2184.0	0.19

Number of instances DPLL couldn't solve based on clause ratio

Number of variables	Clause ratio of 1	Clause ratio of 4.26	Clause ratio of 6
20 (100)	0	44	96
40 (200)	0	50	99
60 (300)	0	45	100
80(400)	0	58	100

Analysis:

Under the clause ratio of 1, the runtimes of all algorithms but Randomized Rounding grow at a slow rate. Randomized Rounding seems to have a high runtime growth rate for any clause ratio. Furthermore, it seems that as the clause ratio increases (less chance of being satisfiable) Randomized Rounding runtime growth increases as well. For DPLL under the clause ratio of 4.26 (each instance is around as likely to be satisfiable as unsatisfiable), there is a big change in the growth rate from the clause ratio of 1. It starts to increase at a very fast rate. As the clause ratio increases from 4.26 the runtime growth becomes low again. GSAT and the Seven Eighths Approximation are very fast no matter the case. (For Graph- blue = Randomized Rounding, red = DPLL)



Satisfiability (average, clause ratio of 1)

# of variables	DPLL	GSAT	Random Rounding	Seven Eighths Approximation
20	1	0.9529	0.9902	0.8828
40	1	0.9359	0.9940	0.8773
60	1	0.9279	0.9931	0.8741
80	1	0.9176	0.9949	0.8790

Satisfiability (average, clause ratio of 4.26)

# of variables	DPLL	GSAT	Random Rounding	Seven Eighths Approximation
20	1	0.9335	0.9003	0.8691
40	1	0.9322	0.8960	0.8754
60	1	0.9232	0.8958	0.8769
80	1	0.9215	0.8957	0.8731

Satisfiability (average, clause ratio of 6.0)

# of variables	DPLL	GSAT	Random Rounding	Seven Eighths Approximation
20	1	0.9325	0.8861	0.8775
40	1	0.9230	0.8858	0.8734
60	NaN (none solved)	0.9150	0.8896	0.8758
80	NaN (none solved)	0.9129	0.8856	0.8737

Satisfiability (Instance size of 400, clause ratio of 1)

# of variables	DPLL	GSAT	Random Rounding	Seven Eighths Approximation
Average	1	0.9176	0.9949	0.8790
Best	1	0.9525	1	0.9125
Worst	1	0.865	0.965	0.8050
Median	1	0.92	1	0.8800

Satisfiability (Instance size of 80, clause ratio of 4.26)

# of variables	DPLL	GSAT	Random Rounding	Seven Eighths Approximation
Average	1	0.9215	0.8957	0.8731
Best	1	0.9617	0.9411	0.9088
Worst	1	0.8705	0.8558	0.8176
Median	1	0.9235	0.8970	0.8735

Analysis (Cont.) :

All approximation algorithms perform well on average. Seven Eighths Approximation gets just about ⅞ each time regardless of instance size or clause ratio. GSAT's results are also pretty constant over instance sizes and clause ratios. However, under individual instances these two algorithms sometimes perform poorly. The worst satisfiability can be much less than their average or median. As for Randomized Rounding, under a clause ratio of 1, its satisfiability was very high for all instances. It was able to satisfy all clauses most of the time (median of 1) and never got less than satisfying 0.96 of all clauses. However, under larger clause ratios, Randomized Rounding performs much poorer. Under a clause ratio of 4.26, it performs worse than GSAT in average,median,best and worst. Additionally, Randomized Rounding takes much longer time to solve instances than GSAT.

TSP:

Algorithms:

These tests record the runtime and optimality of the algorithm, Rosenkrantz, Lewis, and Stearns 2-approximation, on pre-created and pre-solved TSP instances with triangle equality.

Notes:

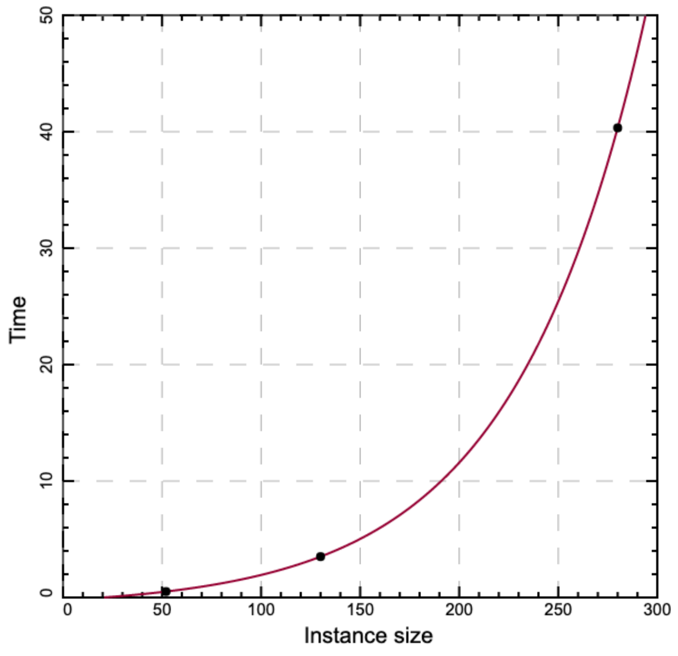
- The sorting algorithm used is Quicksort.
- The algorithm is run 100 times for each instance and then average, best , worst and median are computed for runtime and optimality (solution/optimal solution).
- I attempted to run an instance of size 33000 but was unable to finish the algorithm.

Optimality and Time

	Average optimality	Best optimality	Worst optimality	Median optimality	Average time	Best time	Worst time	Median Time
berlin52.tsp	1.6170	1.4683	1.8026	1.6202	0.51	0.0	1.0	1.0
ch130.tsp	1.6917	1.5517	1.8998	1.6763	3.51	3.0	6.0	3.0
a280.tsp	1.7609	1.5792	1.9054	1.7510	40.33	34.0	120.0	39.0

Analysis:

This approximation is less accurate than many of the ones seen previously. Although it stays within the 2-approximation, it never gets that close to the optimal answer (best seen is 1.46). Furthermore, the worst it can perform is close to double the optimal answer (worst seen is 1.9). As for the time complexity, there is a fast growth. (check out graph below) It would seem that this approximation algorithm would have a hard time solving very large instances. For example, I attempted to solve an instance of size 33000 but was unable to complete it. Additionally, we see that the runtime is not consistent. On the third data set, the worst runtime is three times the average runtime. (running on the same instance!)



Conclusion:

This experiment showed for the NP-hard problems the effectiveness of many approximation algorithms. The algorithms were able to decrease the time complexity for a tradeoff of optimality. Often, this tradeoff is favorable. A large decrease of time is traded for a small error in optimality. The extent of this tradeoff varies from algorithm to algorithm. And so, the best algorithm to use depends on the use case of the situation. For example, in Maximum 0-1 Knapsack if you can afford error in individual instances the 2-approximation is clearly the best. However, if you need to have all instances close to the optimal answer then FPTAS with a high epsilon might be better. In conclusion, by using these algorithms we are able to solve many problems with low error that were seen as practically unsolvable based on its time complexity.