Stefan Dierauf

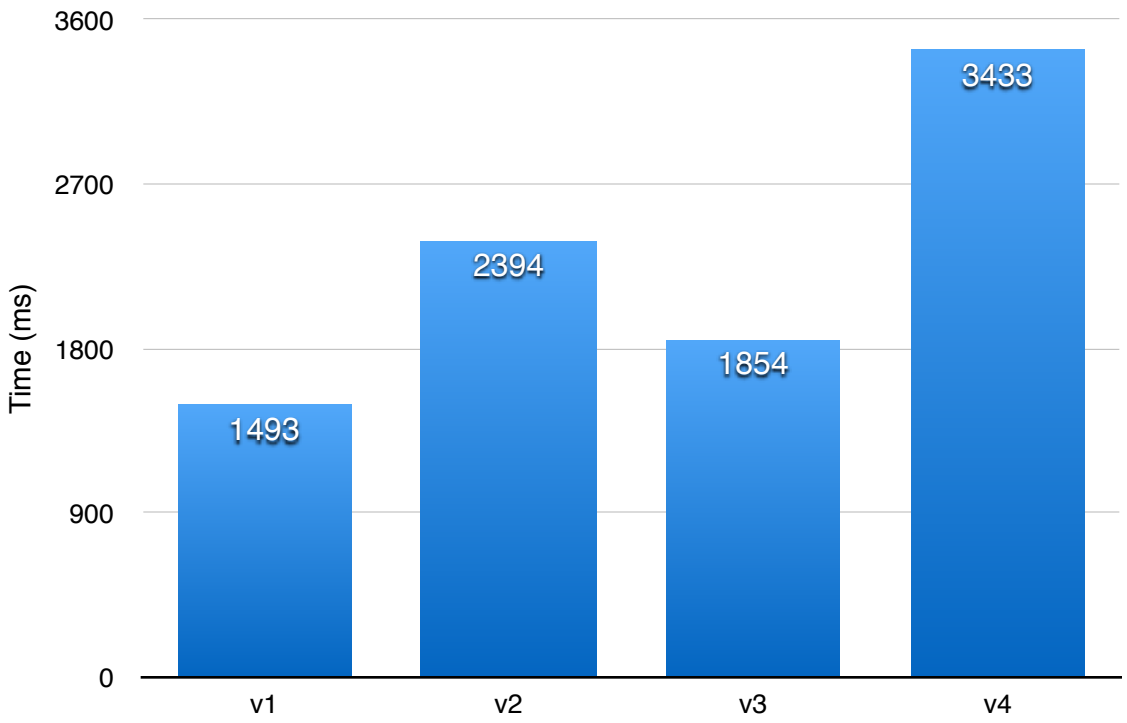
HW6 Experiments

1. My partner was Jack Conger. Jack was very busy with Theater for the first week of the assignment, so I wrote hacky, crappy implementations of all four versions. The week of the 17th was Jack's turn to shine, and he did all the polishing and refactoring. All of the following experiments were run on a 2014 i7 quad core processor with hyper threading
2. I wrote a test suite that had all the example input and outputs provided by The Maestro, including all the following on a 20x25 grid plus two 100x500 tests and one 1x1 test.

   Hawaii: 1 1 5 4: 1360301 .44%
   Alaska: 1 12 9 25: 710231 .23%
   Main US: 9 1 20 13: 310400795 99.34%
   Whole grid: 1 1 20 25: 312471327 100%
   Bottom 4 rows: 1 1 20 4: 36493611 11.68%
   Middle-ish 3 columns: 9 1 11 25: 52392739 16.77%

   Run w/ 2010 census data on grid with 9 columns and 14 rows:
   5 5 7 5: 18820388 6.02%
   6 3 8 4: 105349619 33.71%

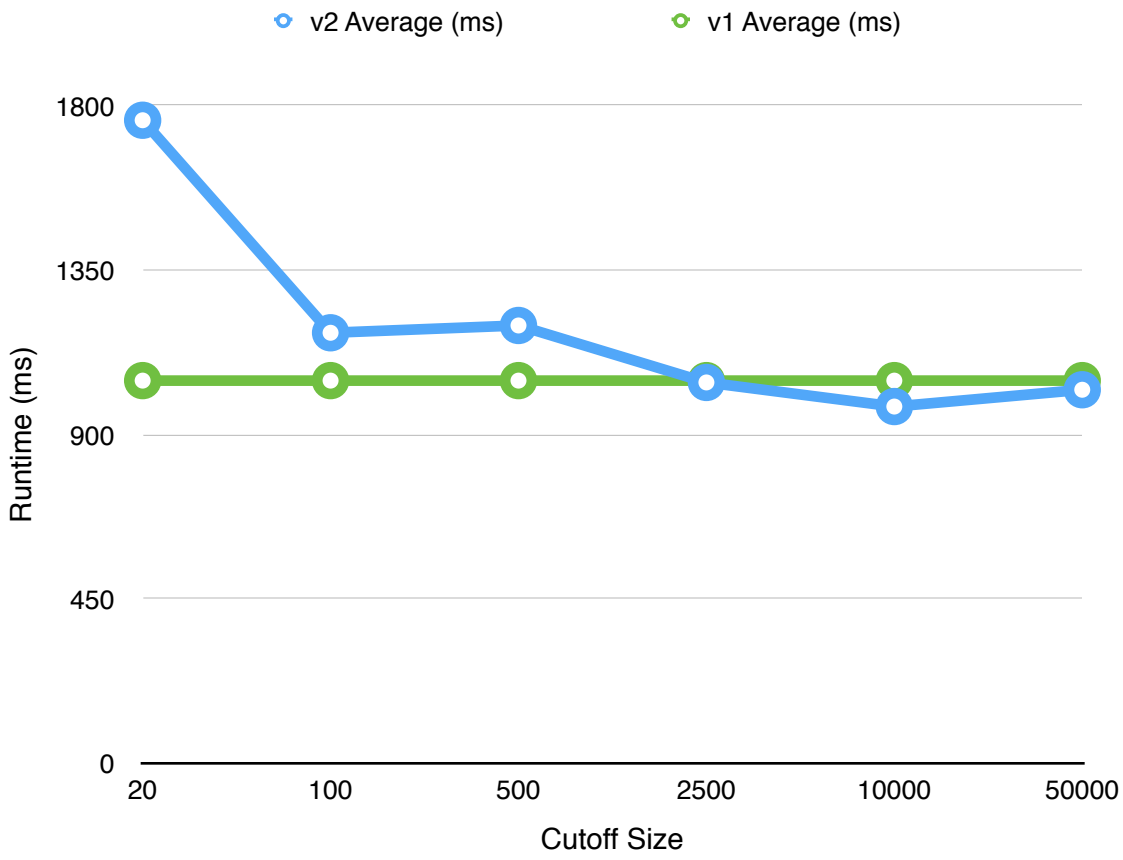
3. For the following results, one run is equivalent to 1) instantiating the solution and then 2)



running a single query on it.
These numbers are each the average of the last 20 of 100 runs for each algorithm. As we

can see, v1 was the fastest, and v3 the second fastest, both of which are not parallel. V3 has a slightly higher cost than v1 probably due to the extra calculation it does to optimize. In this case, parallelization did not help, and would probably not help unless dealing with much larger sets of data.

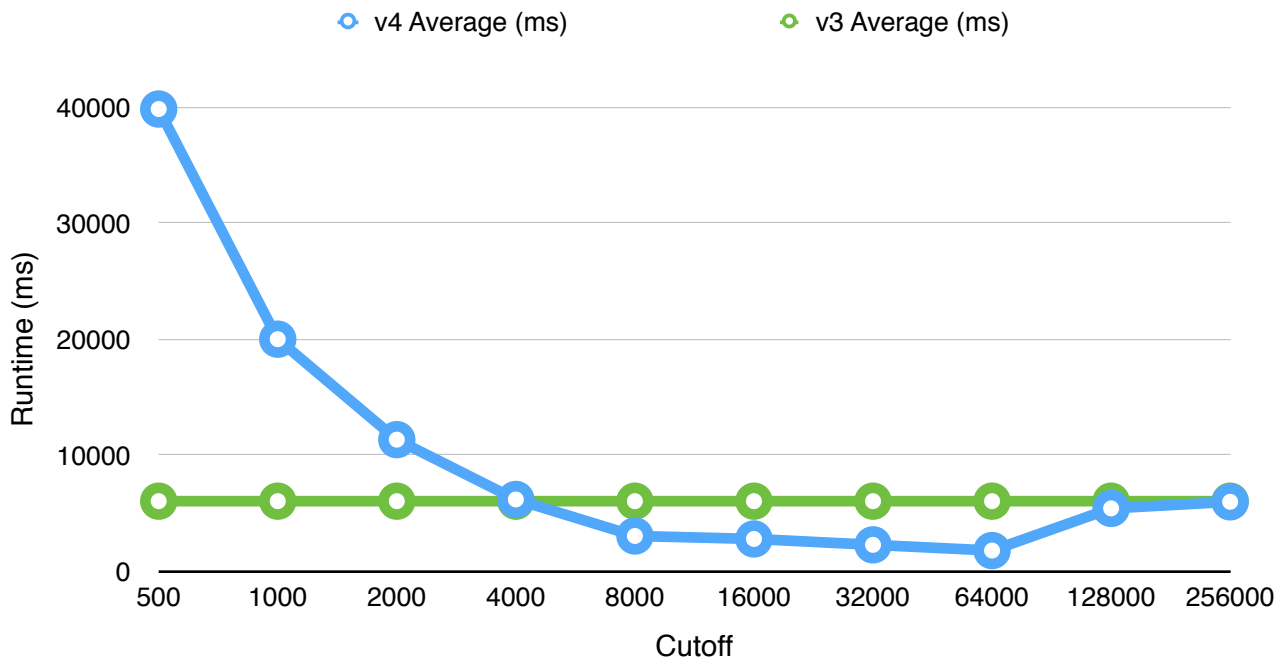
4. Two experiments were run here. First, we have a comparison of version one and version



two,

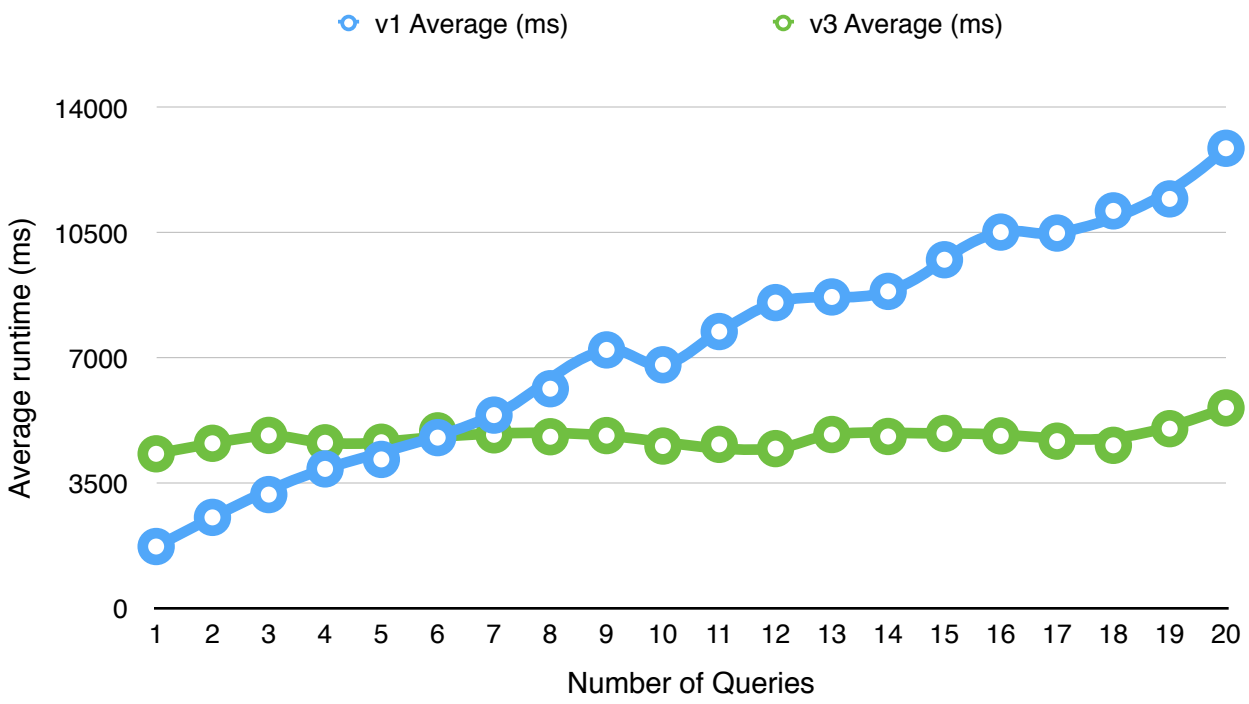
Each result is the average time of the last 8 runs in a series of 20 runs, where a timer was started right before instantiating the array and stopped after instantiation was finished.
As we can see from the data, a small cutoff relative to the data (20 vs 220,000) is far too expensive, but as the cutoff approaches about 1/20th of the array, it becomes worth it. With a cutoff at 1/4 of the size of the data array, it starts matching back up to linear again. Honestly, I think both experiments just ran too fast, and my timer isn't sensitive enough to get a good discernible difference between v2 and v1 except when v2 makes a very large amount of threads

Secondly, we compare version 3 and version 4. Both versions do some optimization to ensure queries complete in constant time. Each point on the graph again represents the average time of the last 10 of 20 runs, with the Timer started right before instantiation and stopped right after.

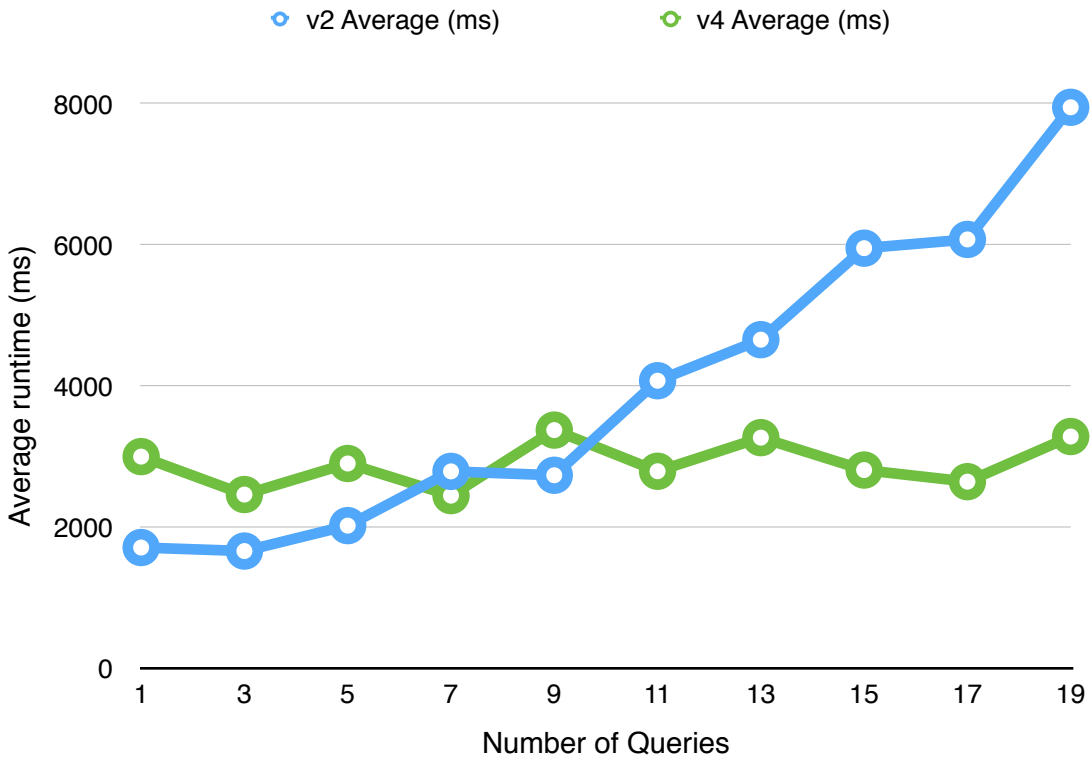


As we can see, with a very low cutoff version 4 takes far longer than linear time. But after a cutoff of 4000, we see that v4 is actually a better choice than v3, becoming best at 64000, about a quarter of the size of the data array. And of course, the linear runtimes match up again as the cutoff approaches the size of the data array.

5. In this experiment, we compare the runtime of multiple queries for each comparable version. Version one and version 3 are both sequential, however version 3 does some optimization on instantiation that enables queries to v3 to run in constant time, whereas version 1 must traverse the whole set of data each query. Looking at the data, it is evident that after around 7 queries version 3's optimizations become worthwhile. Each data point represents the average of the last 10 of 20 runs. (see next page)



I repeated the same experiment for version 2 vs version 4, and saw similar results —it takes



about 10 queries for v4's optimizations to become worthwhile.

6. None attempted

I don’t have every experiment file that I used, as I had to sometimes modify the code for each implementation to add Timers, but I did write a test suite, which I have included.