Esteban Valle
CS 383 Algorithms
Prof. Harmon
7 October, 2015

Esteban Valle's **GrandTur**™
finite-state linear automaton

Runtime Analysis of Turing Machine Simulation

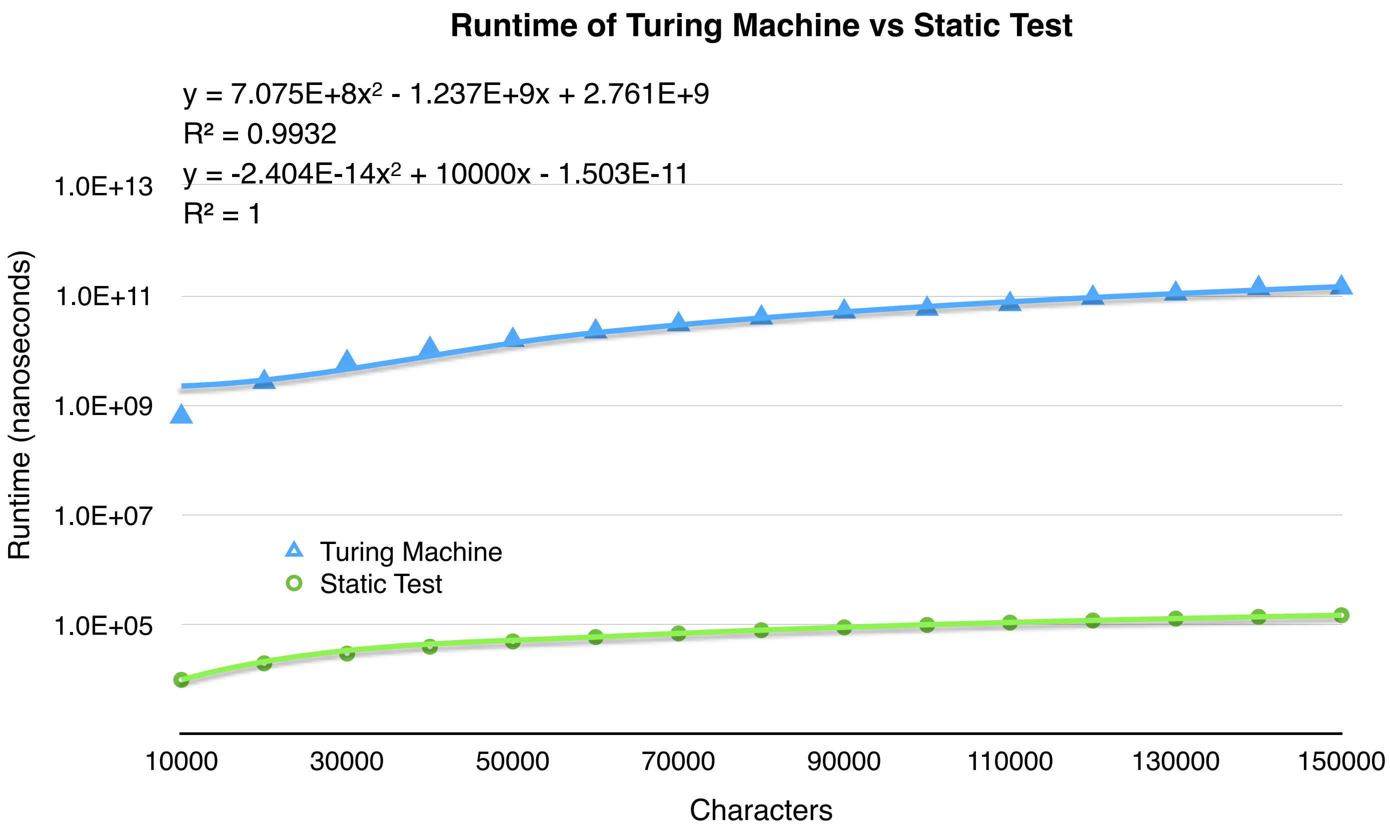

Figure 1: Runtime analysis of the Turing machine compared to the static algorithm revealed that the Turing Machine had, on average, 5 orders of magnitude higher runtime.

The Turing Machine (represented in blue) had a significantly higher runtime (although of the same order) than the static method which accomplished the same task. This is more than likely because the turing machine will have to navigate a huge fraction of the input string before finding the next state to transition to (thousands of moves per step potentially), whereas the static implementation will only have to navigate the string once.

The graph above plots runtime of the Turing machine and linear algorithm against size of the input array of characters, up to 150,000 characters (all tests are accepted by both algorithms). The vertical axis of this graph is logarithmic. The graphs fit to a polynomial (second degree, or quadratic) order, with both the static and Turing implementations accepting a second order polynomial fit line with a high degree of fit ($R^2 > 0.99$). Because the turing runtime is so much larger for a given input than the linear static comparison, a logarithmic graph was the only way to display both on the same axis.

Esteban Valle's **GrandTur**™
finite-state linear automaton

This suggests that although the runtime of the Turing simulation is many orders of magnitude above the linear-time static method (with later tests averaging around three minutes), it does not increase in an unconscionable fashion such as exponential or factorial schedule. I'm not entirely certain that this result makes sense considering the data, because the static method should, ideally, run in linear time. The Turing machine should likely run in time similar to order n*log(n) given the way in which the iterable space decreases with each iteration.

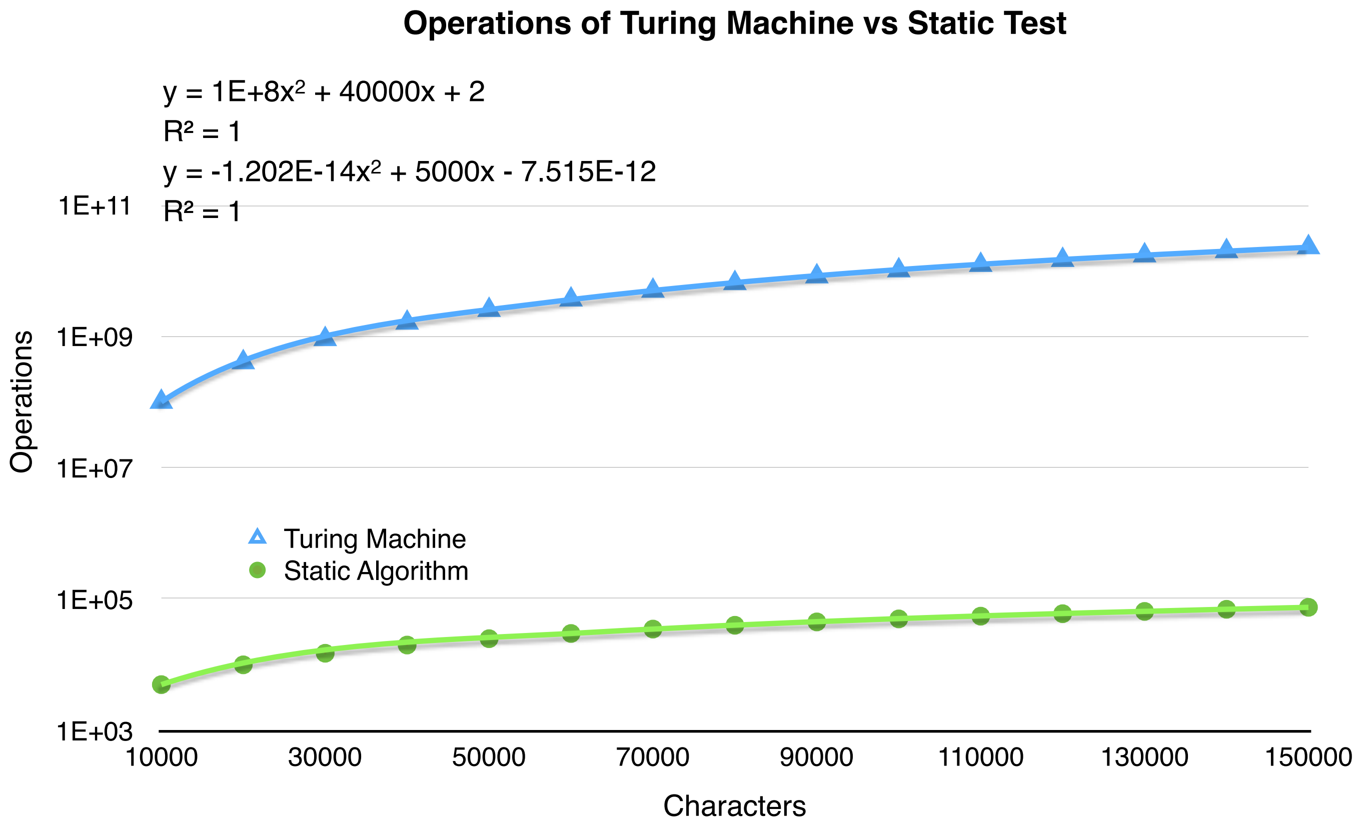

Figure 2: The Turing machine underwent on average 5 orders of magnitude more operations than the linear algorithm. Logarithmic scale reveals a quadratic (2nd degree polynomial) relationship between operations performed and character input length.

Unfortunately, the comparisons are not entirely valid with regards to the actual algorithm in place. This is because the Turing machine has significant additional overhead as compared to the array-based parsing method. Although the timer was started after loading the input onto the Turing tape, the significant overhead of object manipulation, pointer dereferencing, and other sequela of object-oriented Java means that a significant percentage of the time was spent deep within the system managing objects and pointers, rather than running the algorithm.

The entire Turing algorithm could be substantially improved by the simple addition of a cell counter, or some other register which maintained relative position of the head on the Turing tape. With this in place, the Turing machine could easily throw out inputs that aren't even in length, and could iterate much quicker across the tape without having to "overshoot" and come back to the correct cell.

Esteban Valle's GrandTur™ finite-state linear automaton