Final Project Report – Programming Language (CS – 571)

Group 4 - Team members

1. Matthew Reigada
2. Elliot Way
3. Sreedhar Kumar

The individual contributions of the team members are as following:

1. Matt Reigada – Matt worked on the Prolog and Python versions of the Hexadoku solver and worked on the compilation of the report.
2. Elliot Way – Elliot worked on the Java version of the solver and carried out the integration of the functions into Java along with the timing function.
3. Sreedhar Kumar – Sreedhar worked on the C and Javascript versions of the Hexadoku solver and worked on the compilation of the report.

Report:

The Java solver implements an algorithm that prunes a greater portion of the search space while the C, Javascript and Python implement an algorithm that searches a relatively larger space.

Run times for various inputs -: (Time increasing from left to right) (Unit in seconds)

1) **emptyInput.txt**

   C: 0.006 Java: 0.018 Python: 0.214 JS: 0.848 Prolog: 4.188

2) **sparseInput.txt**

   Java: 0.019 C: 0.747 JS: 2.758 Prolog: 4.569 Python: 10.817

3) **hardInput.txt**

   Java: 0.016 Prolog: 1.107 C, JS, Python: N/A – run time exceeds 300.

4) **everyOtherInput.txt**

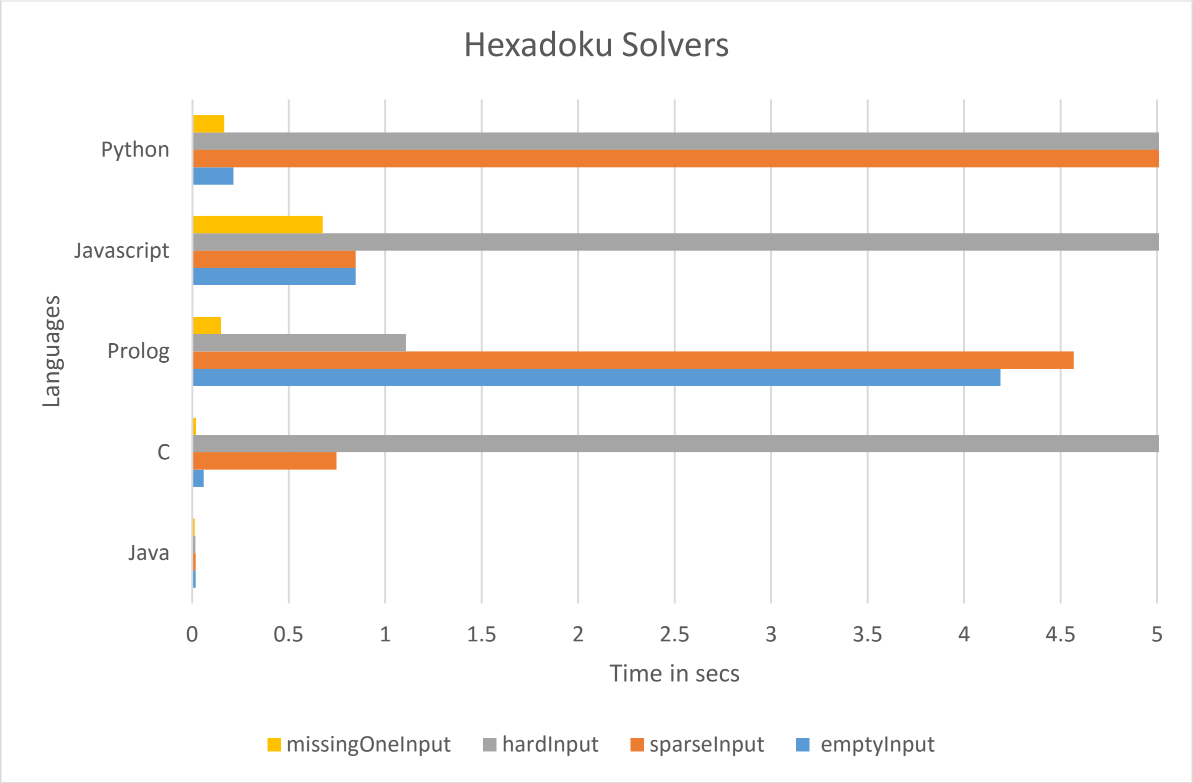
   Java: 0.017 Prolog: 0.553 C: 18.948 JS: 43.048 Python: 290.347

5) **missingOneInput.txt**

   C: 0.002 Java: 0.014 Prolog: 0.149 Python: 0.165 JS: 0 .677

6) **invalidInput.txt**

   C: 0.001 Java: 0.008 Python: 0.128 Prolog: 0.129 JS: 0.630



<u>Effect of language and number of blanks on the run time and performance of the implementations</u>:

With very few blanks all solvers substitute the first available character which does not violate the constraints available within the same row, column, and block spaces. The runtimes seem to follow a bell shaped curve with respect to the number of blanks for the C, Python and Javascript solvers. The Prolog version takes longer to solve the puzzles with more blanks as it does extensive backtracking to arrive at the solution. The Java procedure implements more sophisticated pruning techniques than C, Python, and Javascript, and as a result it does not share the bell shape for execution times with respect to the number of unassigned entries.

With respect to the language used, solvers in interpreted languages like Python and Javascript are relatively slower in coming up with a solution as compared to compiled languages like C and Java. As Prolog is intended for problems of this nature (search problems which require backtracking), a tractable solution is achievable with relatively simple and easily implemented procedures. The Java implementation suggests that imperative languages may yield answers even more efficiently than prolog, but that requires far greater complexity in implementation. The C based solver by itself is faster as compared to the other implementations but lags behind Java because of the difference in pruning techniques used. The Javascript solver runs relatively faster than Python for difficult inputs because of Just-In-Time compilation, while for easy inputs the JIT overhead slows it down.

<u>Citations for the resources used from the internet</u>

1. Java solver - https://github.com/attractivechaos/plb/tree/master/sudoku
2. C, Python and Javascript solvers - http://pythontips.com/2013/09/01/sudoku-solver-in-python/
3. Prolog - http://programmablelife.blogspot.co.at/2012/07/adventures-in-declarative-programming.html