Creating a Determinate Finite Automaton to Solve Self-Avoiding Walks

By Greg Thomas and Caleb Yost

CS454 Final Project #4

Introduction:

A walk refers to taking any path in an n-dimensional grid. In our case n is 2, so there are four different paths a walk can take from any point in a grid. The amount of paths created on a walk depends on distance, k. Four to the 4th power will output the number of paths that can be created in a 2-dimensional grid going k distance. The goal of a self-avoiding walk(saw) is to compute the number of paths, where the path never intersects with itself. After taking a the first walk of distance 1, you can no longer return the same way you came. Options become limited depending on the path the walks take, instead of having 4 possible moves you could have 3, 2, or 1 moves, after the first move. As k increases, the complexity of the problem increases. Through research we were able to find the number of paths created with k, 0-39.

Methods:

In order to create our calculations we began with a DFA, that could calculate a path with memory of the previous 4 walks provided in the Ponitz and Tittmman Paper[1]. Using their DFA we were able to create a transition matrix. With this transition matrix you can use matrix multiplication by multiplying it by itself k times, then multiplying the result by a list of starting states, and then a list of accepting states. In this case all of the states in the DFA are accepting, except the fail state. Walks only go to the fail state once they have taken one of the paths that creates an intersection.

Example of how the computations are done in the code

```
result6 = np.linalg.matrix_power(A6,(linenum))
temp6 = np.dot(ss6,result6)
answer6 = np.dot(temp6,acs6)
```

With 4 walk memory we were able to get the number of walks correct up to a k of 5. Creating a DFA for 6 walk memory proved to be more difficult, since we had to figure out all of the possible states. Once the states were mapped out, we had to use state minimization techniques in order to get rid of states that did the same thing, and we had a 14 state DFA. The same Ponitz and Tittmman Paper[1] helped us compare our states with the states they did for 3 dimensions. Once the transition matrix was created we compared the numbers with the actual numbers. At the a k of 6 we were off by 8. This led us to believe that we possibly had a incorrect transition or minimized the states too much. Comparing the 6 memory walk and the 4 memory walk we were able to create a DFA that has 24 states that is supposed to keep track of a 8 memory walk. The 8 memory walk DFA gave closer results to the accurate value after k was 9 or greater. We also found an equation online thanks to Mireille Bousquet-Melou's powerpoint[2]. We took this equation and edited it in order get the percent error lower for the higher numbers. On lower number of walks the equation produces higher percent error, but as the number of walks gets higher, the percent error gets lower.

Transition table to walks of memory 4:

```
[[0 4 0 0 0]
 [0 1 2 0 1]
 [0 1 1 1 1]
 [0 1 1 0 2]
 [0 0 0 0 4]]
```

Transition table for walks with memory 6:

```
[[0 4 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 2 0 0 0 0 0 0 0 0 0 1]
 [0 0 1 0 0 0 2 0 0 0 0 0 0 1]
 [0 0 0 1 1 1 0 0 0 0 0 0 0 1]
 [0 0 0 1 0 0 0 1 1 0 0 0 0 1]
 [0 0 0 1 0 0 0 1 0 0 0 0 0 2]
 [0 0 0 1 1 0 0 0 0 0 0 1 0 1]
 [0 0 1 0 0 0 1 0 0 0 0 0 1 1]
 [0 0 0 1 1 0 0 0 0 1 0 0 0 1]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 3]
 [0 0 1 0 0 0 0 1 0 0 0 0 0 2]
 [0 0 0 1 0 0 0 0 0 0 1 0 0 2]
 [0 0 0 1 1 0 0 0 0 0 0 0 0 2]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 4]]
```

Transition table for walks with memory of 8:

```
[[0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2]
 [0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2]
 [0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 2]
 [0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2]
 [0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 2]
 [0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 3]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 1]
 [0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4]]
```

Results:

All of our DFA results are able to computer 0% up to when k is 5. The 6-memory DFA and the 8-memory DFA were able to stay between 0-2 percent error for values of k up to 10. When k is 39, the highest value we are able to take, The DFA with 6-memory walk has 165% error, the DFA with 8-memory walk has 106% error, and the equation produces 16% error. Obviously there are plenty of loops we are not accounting for, since loops that can be created with more than 8 memory are not accounted for. For time complexity we are having are multiplying a matrix by itself k times so we are at k*n^3, where n is the length of one of the sides. All of our matrix's are squares so the problem of figuring out the time complexity becomes easier. The starting states and the accepting states are also n each, so we end up with a total time complexity of Big-Oh (2k*n^3).

Sample output:

```
---CS454 Final Project: Self Avoiding Walks---
Enter a number between 0 and 39, in order to compare SAW methods
: 10
Data for the number 10
The number of walks for is 1048576
The correct number self-avoiding walks is 44100
The number of self-avoiding walks using a DFA with memory of 4: 51652
The number of self-avoiding walks using a DFA with memory of 6: 45508
The number of self-avoiding walks using a DFA with memory of 8: 44244
Using the equation that was found and edited: 36038
The percent error of self-avoiding walks using a DFA with memory of 4: 17.124716553287982%
The percent error of self-avoiding walks using a DFA with memory of 6: 3.1927437641723357%
The percent error of self-avoiding walks using a DFA with memory of 8:  0.3265306122448 98%
Percent error using the equation that was found and edited: 18.281179138321995%
```

```
---CS454 Final Project: Self Avoiding Walks---
Enter a number between 0 and 39, in order to compare SAW methods
: 39
Data for the number 39
The number of walks for is 3022314549036572936765 44
The correct number self-avoiding walks is 113101676587853932
The number of self-avoiding walks using a DFA with memory of 4: 660893932171111396
The number of self-avoiding walks using a DFA with memory of 6: 300605760989084532
The number of self-avoiding walks using a DFA with memory of 8: 233720552418337924
Using the equation that was found and edited: 94986774780323456
The percent error of self-avoiding walks using a DFA with memory of 4: 484.3361054491082%
The percent error of self-avoiding walks using a DFA with memory of 6: 165.78364712001695%
The percent error of self-avoiding walks using a DFA with memory of 8:  106.6464083198545%
Percent error using the equation that was found and edited: 16.016475046202675%
```

Future Work:

There is still a lot of work to do when developing these DFAs. We went into this project with the goal of making it so we can produce a delta function that can create DFAs with k-memory. We looked into this by trying to create a function that would capture all of the paths of a walk at length k. We had trouble creating this function because we always ended up with less paths than we wanted. Once a function works, you can compare the paths with others and begin minimizing the states. You can also use the information provided from the function to construct a transition matrix through a delta function.

Works Cited

[1]Pönitz, André, and Peter Tittmann. “Improved Upper Bounds for Self-Avoiding Walks in ${\Bf Z}^{d}$.” *A Survey of Venn Diagrams: What Is a Venn Diagram?*, www.combinatorics.org/ojs/index.php/eljc/article/view/v7i1r21/pdf.

[2]Bousquet-Melou, Mireille. “Two-Dimensional Self-Avoiding Walks.” *CNRS*, www.labri.fr/perso/bousquet/Exposes/fpsac-saw.pdf.

[3]Conway, A R, et al. *Algebraic Techniques for Enumerating Self-Avoiding Walks on the Square Lattice*. Journal of Physics A, 2018, arxiv.org/pdf/hep-lat/9211062.pdf.