Physics of Algorithms Final Report

Finding De Bruijn Sequences using Simulated Annealing

Yifan Liu

April 21, 2021

1 Introduction

1.1 Background

The idea of this report came from a blog article [1] that I came across a long time ago which I read as entertainment material. As you might have guessed it was about a mathematical concept namely De Bruijn Sequences.

Basically "a de Bruijn sequence of order n on a size-k alphabet A is a cyclic sequence in which every possible length-n string on A occurs exactly once as a sub-string" [2].

1.2 Trigger

When my friend moved in to a student dormitory. I was also able to get access to the dormitory through the key code system. The system only looks at the last 4 digits of code input by the user and if it is a match the door opens. For example, if the password is 3456 and the input is "12233456" the door will open because the last four digits matches the password. This reminded me of the de Bruijn sequence that was known to be used to crack a key code system like this with brutal force.

In this case, we are talking about a de Bruijn sequence of order 4 and an alphabet size of 10 (0-9). It happens to contain every possible 4 digit combination and therefore the door is guaranteed to open. The size of this de Bruijn sequence is said to be 10000 cyclic digits, meaning the end of the sequence is connected to the beginning. From this we can get a non-cyclic sequence of 10003 digits that contains every possible code combination.

Although there are multiple established ways to construct a de Bruijn sequence of any order and alphabet size, in this report, I am going to explore the possibility to construct a de Bruijn sequence using simulated annealing algorithm.

2 Setup of Simulated Annealing

For simulated annealing to work there are some essential elements, Below, I will use a de Bruijn sequence of alphabet size of 10 and order 3 to illustrate the definition of these elements.

2.1 Starting State

First randomly take 334 sequences from the 000-999 space and put them together head to tail to get a sequence of length 1002(just enough for 1000 unique sub sequences or order 3). However, because de Bruijn Sequence has the property to wrap around, hence in this case the last 2 digits of the sequence should be the same as the first 2 digit of the sequence. The relation between the first sub sequence and the last sub sequence should be obeyed when constructing the starting state. For example, 467,234,216,…,886,456,246 —> 467234216 … 886456246. Now the sequence we just constructed has the possibility to be a de Bruijn sequence, to contain 1000 unique sub sequences, this wrap around relation is followed because de Bruijn Sequence is the known global minimum of the system.

2.2 Operations towards a neighboring state

Now with the starting state I then choose the first sub sequence of the 334 sequences that construct the main sequence and replace it with a random sequence taken from the 000-999 space space. If the replacement sequence is already one of the 334 sequences then I simply switch their position within the 334 sequences. And if the operation results in a change of the first 2 digits of the main sequence then I change the last sub sequence too(randomly), in a way that satisfy the wrap around condition. However when this happens two sequences are changed at the same time, resulting in a neighboring state that is not exactly the closest neighbor, the same goes for when two sub sequences change their position.

Another way of ensuring the cyclic feature of the sequence and at the same time a better way to move towards a neighboring state, when the first sequence is changed, than previously described is to simply copy the first (n-1) digits of the first sequence to the end of the sequence. This only works if the number of unique sub sequences are divisible by the order of the sequence. One case would be for a de Bruijn sequence of order 3 and a alphabet size of 6. In this case, there are $6^3 = 216$ unique sub sequences, the minimum length of the main sequence that contains all sub sequences is 218. Therefore, 72 unique sub sequences will be chosen to construct the main sequence which is of length 216 and conveniently the last two digits can be directly copied over from the first two digits to make the main sequence of length 218. However this way of ensuring cyclic feature only works for certain alphabet size and order combinations that fulfill the previously described constrains.

2.3 Energy

The Energy of the state E is then the total number of unique sub sequences minus how many unique sub sequences is contained in the main sequence. E = 1000-800 = 200, if there are 800 unique sub sequences found in the main sequence and E=0, if all 1000 unique sub sequences are in the main sequence, hence by definition a de Bruijn Sequence. Change in Energy(dE) is the new Energy minus the old Energy.

2.4 Temperature and Probability function

The Probability function is defined as follow for dE lager than 0, all dE smaller than 0 is accepted right away.

$$P = e^{-\frac{dE}{T}} \tag{1}$$

where temperature function T is defined as follow, with starting temperature denoted as T_0:

$$T = T_0(e^{\frac{1}{t-1}} - 1) \tag{2}$$

Time t is defined as a counter for how many operations has been performed.

3 Results

For simplicity all the results are done with an alphabet size of 6 (0,1,2,3,4,5) and order 3.

Before we get into the results let us take a look at what impact different methods of reaching a neighboring state have to the result of the search shown in fig.1. The three categories named in the legend as Unique, Consecutive and Neighbor are constrains to the operation towards a neighboring state.

When Uniqueness Constrain is on, if the replacing sub sequence is already used to construct the main sequence then the chosen sub sequence is to change position with the exiting replacing sub sequence, when uniqueness constrain is off, the chosen sub sequence is just replaced by the replacing sub sequence even though it would result in repeated sub sequences.

When Consecutive Constrain is on it means the Choice of sub sequences to be replaced is chosen consecutively from the 1st to 2nd, all the way to the end to make sure every sub sequence is operated on equally many times. When Consecutive Constrain is off, it is chosen randomly within the sub sequences that construct the main sequence. This constrain is left "on" the whole time.

When Neighbor Constrain is close, if xxx is chosen to be the replaced sequence it is replaced by xxx+1 or xxx-1 instead of a random choice xyz, when Neighbor Constrain is random.

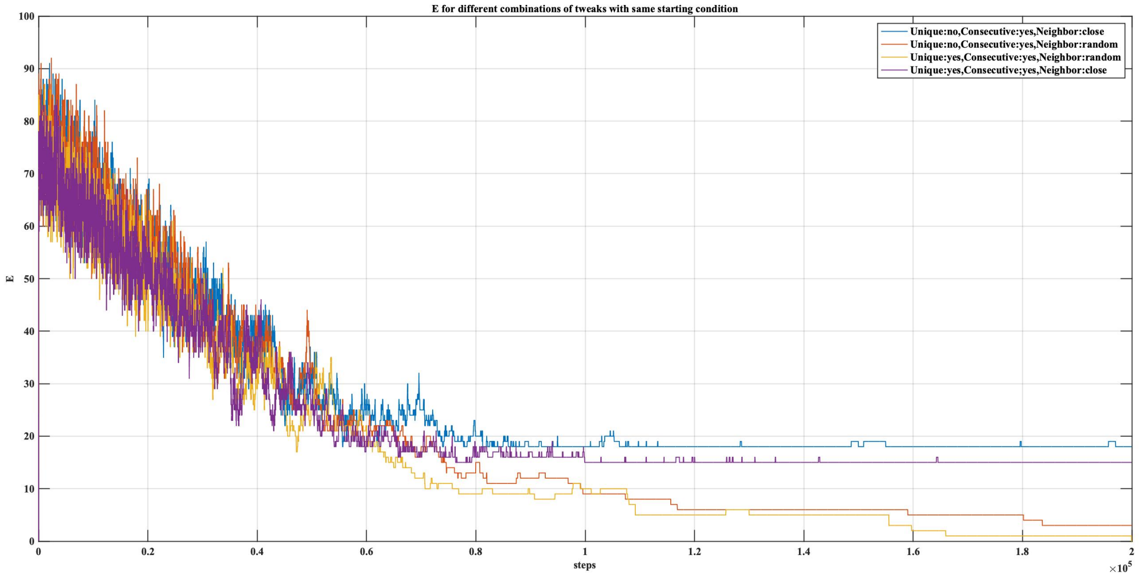

Figure 1: Uniqueness constrain and random neighbor selection generally helps to get a lower E. A combination of both resulted in the lowest E of all.

From fig.1 we can see that Uniqueness Constrain "on" and Neighbor Constrain "random" generally helps to get a lower E, so that is what we will use for the rest of the report.

The results of the search using simulated annealing is shown in fig.2.

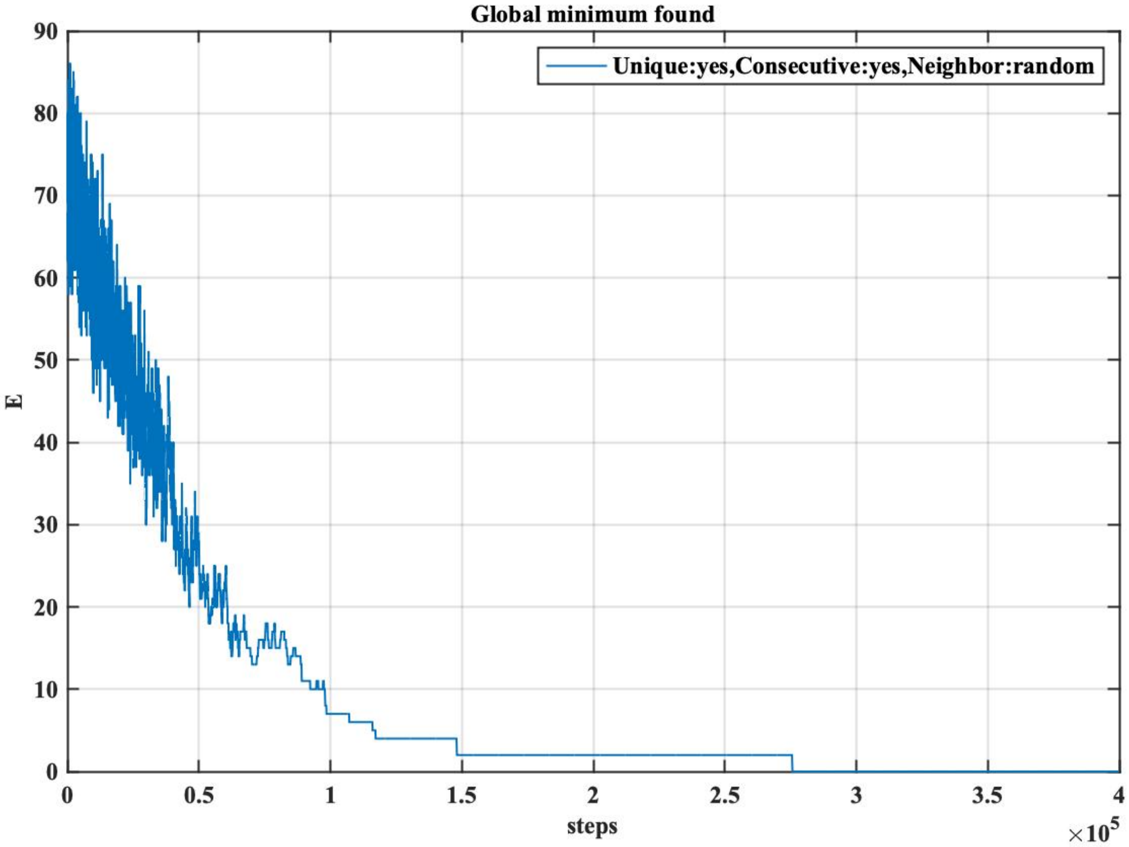

Figure 2: The search got to E=0 after about 2.75E5 steps.

A comparison of greedy search and simulated annealing method on the same starting condition is shown in fig.3.

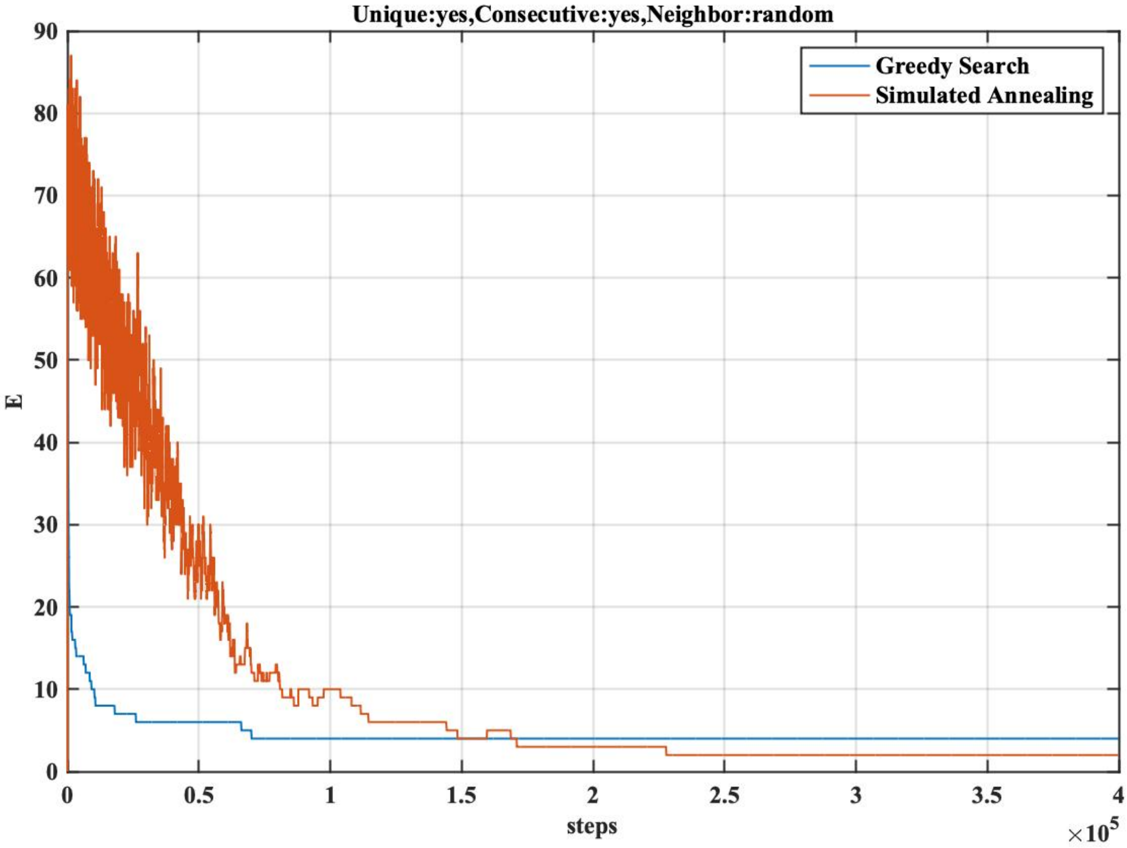

Figure 3: Although non of the search reached E=0 in this perticular case, we can see that simulated annealing helps to get a lower E than greedy search method.

The starting temperature also plays an important role in simulated annealing. As shown in fig.4. A starting temperature of 20000 yields a simulated annealing that got stuck at a local minimum around E = 2 while a starting temperature of 30000 yields a simulated annealing search that was able to reach the global minimum at E = 0.

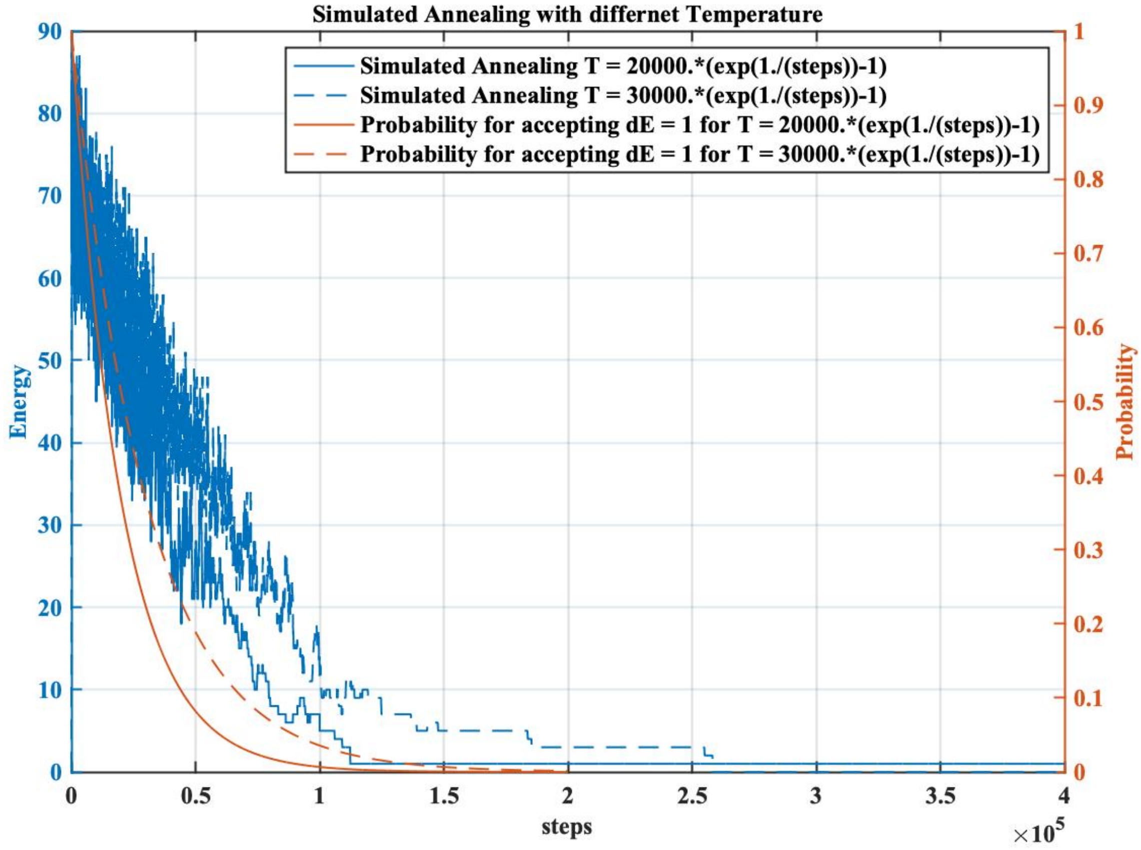

Figure 4: Effects of different starting temperatures in simulated annealing.

4 Discussion

The obtained results used a combination of Uniqueness Constrain and random Neighbor Constrain to move towards a neighboring state. Uniqueness Constrain makes sure that the main sequence has a potential to be a De Bruijn Sequence at any given moment. Random Neighbor Constrain makes the "search" easier by allowing the neighboring state to span cross the phase space instead of limiting to the xxx+1 and xxx-1 neighboring state. This is apparently good for shortening the time for the search of global minimum in a large phase space as shown in fig.1.

5 Conclusion

This report shows that for a relatively small alphabet size and order, simulated annealing method is a viable, faster way to finding a De Bruijn Sequence when compared to greedy search.

References

[1] Nick Berry, *De Bruijn Sequences*, `http://datagenetics.com/blog/october22013/index.html` Last accessed, July 2020.

[2] *De Bruijn Sequences - Wikipedia*, `https://en.wikipedia.org/wiki/De_Bruijn_sequence` Last accessed, July 2020.