Solution of the Hamiltonian Path Problem:  
A Sequential Simulation Inspired by DNA Computing  
Versus a Classical Method  

Cartik R Kothari, David J. Russomanno, and Russell J. Deaton

1Department of Electrical and Computer Engineering  
The University of Memphis  
Memphis, TN 38152 USA  
{rkothari, d-russomanno}@memphis.edu  

2Department of Computer Science and Computer Engineering  
University of Arkansas  
Fayetteville, AR 72701 USA  
rdeaton@engr.uark.edu  

Abstract. In 1994, Adleman demonstrated the solution of a Hamiltonian path problem via a 7-vertex directed graph using DNA molecules. A year later, Lipton generalized Adleman’s method and proposed a method to solve Nondeterministic Polynomial (NP) complete problems by using DNA molecules, with an emphasis on the satisfiability problem of Boolean formulas. Since then, the application of DNA molecules to solve extremely hard computational problems is a subject of intense research with potential applications in a number of areas such as molecular biology, pharmacy and medicine. The purpose of this paper is to present baseline software implementations and subsequent empirical evaluations to compare a classical method versus a sequential technique, which was inspired by DNA computation as a problem solving metaphor.

For the application, two alternate approaches to the solution of the classical Hamiltonian Path (HP) problem are implemented using the PROLOG language. For the first approach, declarative PROLOG queries are used to implement a classical solution to the HP problem. In the second case, a sequential simulation in PROLOG, using DNA-computation as a metaphor, is implemented to solve the HP problem. After successful implementation and testing of both approaches, the efficiency of both is examined by varying the complexity of the problem with respect to the number of nodes and edges involved. A statistical analysis of the collected data is then performed. The classical approach outperformed the DNA-inspired approach. However, the study did not exploit the massive parallelism that is inherent in DNA since a sequential processing realization was used for the later simulation. By implementing the DNA-inspired approach in a sequential manner, the advantages of the massive parallelism of DNA were not achieved; however, the results provide a baseline for subsequent work using a parallel implementation of PROLOG and accompanying hardware to exploit the massive parallelism inherent in the DNA problem-solving metaphor.
1. Introduction

The objective of this paper is to use Adleman’s experiment [1] as a metaphor for implementing a DNA-inspired sequential computation technique and compare its efficiency, based on computation time, to that of a classical technique. The motivation is to provide a baseline result before a parallel-processing approach to the simulation is pursued.

The graphical representation of a typical HP problem is shown in Fig. 1. The Hamiltonian path will be one that visits every node in the graph in Fig. 1 exactly once. A visual examination reveals that a path covering the nodes 1, 2, 4, 5 and 3, in that order, is a Hamiltonian path and is the solution to the problem represented in Fig. 1. If the number of nodes in this problem were significantly increased, more than a cursory visual examination would be needed to find the Hamiltonian path. This is the inherent nature of a typical NP complete problem, the degree of difficulty involved in the solution increases exponentially with the complexity.

![Fig. 1. A graphical representation of a typical Hamiltonian Path (HP) problem with 5 nodes.](image)

The use of sequential computation techniques to solve this problem would run into efficiency problems with increasing complexity. The shortcomings of a sequential algorithm would be exposed in a scenario in which the number of edges and nodes relevant to a problem are scaled beyond a certain threshold. The assignment of separate segments of the algorithm to different parts of a coordinated parallel-processing architecture would reduce the shortcomings associated with a single sequential processor working on the same problem.

Adleman’s Experiment

In 1994, Adleman [1] solved a HP problem involving 7 nodes in his laboratory. He represented each node by a unique combination of nucleotides: Adenine (A), Thiamine (T), Guanine (G) and Cytosine (C). In Fig. 1 for instance, the sequence [A, G, T, C] can be used to represent node 1, [A, T, G, C] can represent node 2, and so on. Adleman synthesized each of these unique nucleotide combinations in the laboratory. Then, under set conditions, these nucleotide combinations were hybridized to generate longer strands of varying lengths. This procedure resulted in a large multitude of strands, representing paths in the graph, being generated simultaneously. Strands were then examined *in vitro* by various techniques to find the ones that represented the solution to the problem. The elimination process was approximately as follows: For a problem involving 7 nodes, each of which was represented by a combination of 4 nucleotides, it can be readily inferred that the correct strand representing the Hamiltonian path would have exactly 28 nucleotides. So Adleman filtered out all those strands that did not have exactly 28 nucleotides.

Of the remaining strands, all those in which there was a recurrence of a 4-nucleotide combination were eliminated. Finally, every remaining strand was examined for valid edges (combinations of 8 nucleotides) and only those that comprised a sequence of valid edges were retained. These retained strands represented the Hamiltonian paths, the solutions to the problem.
A computational equivalent to Adleman’s experiment would require parallel processing. The generation of a multitude of hybridized strands simultaneously is the salient feature of Adleman’s experiment, which is impossible to simulate without the use of parallel processing.

2. Implementation

In this section, the two alternate methodologies that have been implemented in this project to solve a classical NP complete problem are discussed in detail. When necessary, examples have been used to elucidate certain points.

**Classical Method**

A flowchart depicting the process that is implemented in this method is illustrated in Fig. 2. The algorithm starts at any of the nodes at random, say node 1. Given the set of edges, it moves to the next node (node 2) using the edge from node 1 to node 2. Here the algorithm checks to see if all the nodes have been visited. Then it looks for an edge that will take it to a previously unvisited node. Finding the edge from node 2 to node 3, the algorithm arrives at node 3. Here again, it determines that two more nodes need to be visited. Furthermore, it finds out that only one edge is available from node 3, taking it back to node 2 that has already been visited. So the algorithm backtracks to node 2 and selects the edge going to node 4 to reach node 4. Again, it checks to see that two more nodes need to be visited and finding the edge from node 4 to node 5, arrives at node 5. At node 5, the algorithm uses the edge going to node 3 to reach node 3. Here it discovers that all the nodes have been visited and terminates with the solution [1,2,4,5,3] to the problem represented in Fig. 1.

![Flowchart representation of the classical algorithm that is used to find the solution to the NP complete problem.](image-url)
Sequential simulation of Adleman’s experiment

Fig. 3 illustrates the flowchart representation of this method. The algorithm that simulates Adleman’s experiment is simplified with respect to the actual experiment in two ways. Whereas Adleman synthesized strands of different lengths simultaneously in the laboratory, the algorithm implemented in this project restricts itself to handling strands of only a specific length i.e., only strands whose length is the same as the one that represents the Hamiltonian path. In the problem represented in Fig. 1, if each node were to be represented by a unique combination of 4 nucleotides, the strand representing the Hamiltonian path comprises exactly 20 nucleotides. The DNA computing simulation algorithm handles only those strands that comprise exactly 20 nucleotides.

Secondly, Adleman had to eliminate all those hybridized strands that had repetitions of specific nucleotide sequences in them. In this algorithm, generating only strands without any nucleotide sequence repetitions has circumvented this problem.

The algorithm that simulates Adleman’s experiment would go about finding a solution to the Hamiltonian path by first assigning unique sequences of nucleotides to each node in the problem. The rules to do this are stated in the data section of the algorithm. The algorithm then comes up with a combination of all these sequences to make up one strand.

Fig. 3. An algorithm that simulates Adleman’s [1] experiment to solve the Hamiltonian Path problem.
If node 1 were represented by [a,g,t,c], node 2 by [t,a,g,c], node 3 by [c,t,a,g], node 4 by [g,c,t,a] and node 5 by [a,c,t,g], the algorithm would come up with a strand that would be for instance, [[a,g,t,c], [t,a,g,c], [c,t,a,g], [g,c,t,a], [a,c,t,g]]. The algorithm then checks to see if the sequence of nodes represented by this strand ([1,2,3,4,5]), is a solution. The path [1,2,3,4,5] is a solution if it is possible to go from node 1 to node 2, then from node 2 to node 3, then from node 3 to node 4 and finally, from node 4 to node 5. The specifications of the problem, however, do not include an edge that goes from node 3 to node 4. The path [1,2,3,4,5] (or its nucleotide sequence equivalent) includes an invalid edge, and as such, is rejected.

The algorithm then generates a permutation of this path, say [[a,g,t,c], [t,a,g,c], [g,c,t,a], [a,c,t,g], [c,t,a,g]]. This is actually the path [1,2,4,5,3]. Given the specifications of this problem, the algorithm verifies that it is indeed possible to traverse all the nodes in this order given the existing set of edges and returns this path [1,2,4,5,3] as a solution to the given problem. The algorithm checks permutation after permutation to come up with a solution. Once it exhausts permutations to test, the problem does not have a solution.

Both these methods are implemented using algorithms coded in SWI-PROLOG [2]. The only aspect of the two algorithms that is taken up for comparison is the computation time involved in arriving at the solution. Comparison of memory utilization aspects of the two algorithms can be taken up subsequently, but is not examined here.

3. Results

The execution of the classical algorithm on a problem comprising 7 nodes and 13 edges used 117 inferences carried out in a computation time of 0.008 seconds. The DNA computing simulation algorithm used 509 inferences carried out in 0.80 seconds to solve the same problem.

On problems of higher complexity i.e., involving more nodes and edges, the classical algorithm consistently outperformed the DNA computing simulation algorithm in terms of computation time and number of inferences used. On a problem comprising 10 nodes and 18 edges for instance, the classical algorithm used 10,485 inferences carried out in 0.03 seconds. The DNA computing simulation algorithm used more than 11 million inferences in 67.0 seconds to solve the same problem. Problems of further increasing complexities only exaggerated this mismatch between the two approaches.

Figs. 4a and 4b graphically represent the computation time of the two algorithms on problems comprising 7 nodes and higher. The graphs in Fig. 4 indicate the classical algorithm outperforming the DNA computing simulation algorithm on all counts. The efficiency of the DNA computing simulation algorithm drops off sharply when the complexity of the problem is scaled beyond 15 nodes. For a problem comprising 20 nodes, the DNA computing simulation algorithm required more than 28 hours without finding a solution. The classical algorithm, in comparison, came up with a solution in only 3 seconds.

The classical algorithm however, ran into problems when the complexity was scaled above 30 nodes. It used up more than 32 hours without finding a solution to a problem comprising 30 nodes and 205 edges.

The general trend of decreasing computation times with increasing number of edges (given the number of nodes stays the same) can be visually inferred from the graphs in Figs. 4a and 4b. This is because both algorithms have more valid edges with the same number of nodes to try and solve the problem. This increasing possibility of finding valid edges is responsible for the decreasing trend in computation times.

In summary, the results obtained from this study indicate shortcomings in the use of Adleman’s experiment [1] as a simulation metaphor without employing parallel processing to solve a typical NP complete problem. The possible reasons for this drawback will be examined in the following section.
Fig. 4a. Comparison of Computation Times for a 7-node problem.

Fig. 4b. Comparison of Computation Times for a 9-node problem.
4. Discussion

The results of this project discussed in the previous section clearly indicate that the classical algorithm outperforms the DNA computing simulation algorithm. A discussion of the implementation features of the two algorithms serves to explain this fact.

The simulation method has not been implemented on a platform with inherent parallel processing capabilities, but rather on one that is only sequentially-capable of DNA computation. As such, instead of a multitude of possible solutions being generated at once (in the form of strands, as in Adleman’s experiment), the DNA computing simulation algorithm generates possible solutions one at a time. This leads to a computation time overhead that far exceeds that of the original experiment. This drawback can be overcome by executing the simulation algorithm, in contiguous segments, on different sections of a coordinated parallel processing architecture. This would make the comparison between the two algorithms (simulation and classical) a more objective one, in terms of computation time.

The classical algorithm uses a bottom-up approach to solving the HP problem. It constructs a solution from scratch, starting off from any node at random, and working towards the solution constraining itself within the specified restrictions. The DNA computing simulation algorithm initially looks up a database and uses it to transcribe every node into its equivalent nucleotide sequence. Then, it uses another set of rules to generate longer strands comprising permutations of these nucleotide sequences. It should be noted that every sequence occurs once and exactly once in these longer strands. Were this condition not imposed, the comparison of computation times between the two algorithms would be far more skewed in favor of the classical approach. The DNA computing simulation algorithm then tests the validity of each generated possible solution to see if it is a solution to the problem. On finding a valid solution, it transcribes this combination of nucleotide sequences back to a combination of nodes by looking up a database.

From this discussion, it can be inferred that the DNA computing simulation algorithm has to make additional logical inferences as compared to the classical algorithm to find a solution. This is another reason for the relative inefficiency of the DNA computing simulation algorithm. It may be possible to enhance the efficiency of the DNA computing simulation algorithm by reducing the inferences required for its execution. This may involve research into programming aspects that cannot be discussed within the scope of this paper. The authors make no claim that a more efficient DNA computing algorithm cannot be implemented.

5. Conclusions and Future Directions

The results of this study expose the under performance of a DNA computing simulation using Adleman’s experiment as a metaphor when it is implemented without parallel processing capabilities. It is evident that the DNA computing simulation technique (or any more efficient equivalent that may be developed subsequently) needs to be augmented with the capabilities of parallel processing in order to improve its efficiency and realize its rich potential. These results can however serve as a baseline for future experiments, involving parallel processing, that can be taken up in this area. The work by Gregory [3] is especially relevant in the context of utilizing a declarative programming language such as PROLOG to implement the simulation. The AND-OR parallelism inherent in PROLOG can also be exploited to this effect as described by Dutra [4,5]. Gupta [6] elucidates a re-computational approach to exploiting the AND-OR parallelism in PROLOG in his work. Other interesting work involving parallel logic having potential applicability to implementing a DNA-inspired parallel approach includes those by Costa [7], Kale [8] and Kacsuk [9]. Procedural and functional techniques could exploit the techniques outlined in the work by Kale [10]. Techniques that involve interfacing a PROLOG algorithm with other programming languages to implement a multi-processor based parallel processing environment include the work by Szeredi [11].

An iterative approach involving generation of partial solutions and subsequent removal of those that cannot be extended into a full solution has been proposed by Suyama [13]. These are some of the new techniques the authors plan to pursue in implementing a simulation study of the efficiency of DNA computation with its inherent parallel processing capabilities.
6. References