The Complexity and Viability of DNA Computations (Extended draft) CTAG-97001

Martyn Amos Alan Gibbons Paul E. Dunne
Department of Computer Science, University of Liverpool, L69 3BX, England

Abstract

In this paper we examine complexity issues in DNA computation. We believe that these issues are paramount in the search for so-called “killer applications”, that is, applications of DNA computation that would establish the superiority of this paradigm over others in particular domains. An assured future for DNA computation can only be established through the discovery of such applications. We demonstrate that current measures of complexity fall short of reality. Consequently, we define a more realistic model, a so-called *strong* model of computation which provides better estimates of the resources required by DNA algorithms. We also compare the complexities of published algorithms within this new model and the weaker, extant model which is commonly (often implicitly) assumed.

1 Introduction

Following the inital promise and enthusiastic response to Adleman's seminal work [1] in DNA computation, progress towards the realisation of worthwhile computations in the laboratory has become stalled. One reason for this is that the computational paradigm employed by Adleman, and generalised by the theoretical work of others [13, 16], relies upon filtering techniques to isolate solutions to a problem from an *exponentially sized* initial solution of DNA. This volume arises because all possible candidate solutions have to be encoded in the initial solution. As Hartmanis points out in [12], the consequence is that, although laboratory computations should work for the smallest problem sizes, the experiments do not realistically scale because vast amounts of DNA are required to initiate computations with even modest problem size. For example, Hartmanis shows that a mass of DNA greater than that of the earth would be required to solve a 200-city instance of the Hamiltonian Path Problem.

If practitioners of DNA computation insist on this mode of computation, there can be no hope of discovering so-called *killer applications*, that is, applications of DNA computation that would establish the superiority of this paradigm over others in particular domains. An assured future for DNA computation can only be established through the discovery of such applications.

It is not inherently the case that exponentially sized volumes of DNA need be used in DNA computation. Indeed, polynomially sized computations have been (at least in theory) described (e.g., in [14]). Clearly, if exponentially sized volumes are to be avoided, then an alternative algorithmic paradigm to that employed by Adleman in [1] is required. Such a successful paradigm is always likely to emulate traditional computations which *construct* individual solutions rather than sift them out of a vast reservoir of candidates. It might still be argued that the "exponential-curse" could not, even then, be avoided for the so-called NP-complete problems [9]. If an exact solution is required for any of these, then (employing any extant algorithm) exponential sequential running time is required. A DNA computation, in seeking to reduce this to sub-exponential parallel running time, will certainly require an exponential volume of DNA. However, in general, no-one sensibly seeks exact solutions to the NP-complete problems. In traditional computation, we either employ heuristics to obtain approximate answers or use randomised methods to obtain exact solutions with high probability. These revised algorithmics lead to solutions within polynomial sequential time. Such a view should also be taken for these problems within DNA computation, that is, we should use algorithms which do not inherently require exponential resources.

It is unlikely to be enough, in the quest for killer applications, to simply have polynomial-volumed computations. We ought, at the same time, to ensure that the vast potential for parallelism is employed to obtain rapid computations. The view taken by the silicon-based parallel computing community [10] is that efficient parallel algorithms, within the so-called Parallel Random Access

Machine (P-RAM) model of computation, should have polylogarithmic running time (and use a polynomial number of processors). Problems for which such solutions exist define the complexity class NC. If DNA computation is to compete within this domain, then we should clearly also look for polylogarithmic running times within polynomially-volumed computations.

At the present time, no-one has described (even theoretically) DNA computations which run in polylogarithmic time using a polynomial volume of DNA. The discovery of such solutions might well provide candidates for "killer applications". Regardless of the problem considered, it is unlikely to provide a "killer application" unless the computational resources required for a DNA computation (the product of the running time and volume of DNA required) match those needed for a conventional computation (the product of the running time and the number of processors used). For such a combination of resources, the DNA computation might well provide feasible solutions for problem sizes far greater than can be achieved by conventional computation.

It is clearly crucial, especially when judging the candidacy of a proposed DNA computation for the role of "killer application", to have a firm grasp of the computational resources that it requires. In this paper we review claims that have been made concerning the complexity of DNA algorithms. We conclude that these claims are often unrealistic, or simply not true. It also the case that there is not an agreed model of computation in the literature within which we may agree what the required resources are for any particular computation. This paper attempts to address these issues in a realistic way.

Traditional computational complexity theory [2, 9] is concerned with quantifying the resources (generally *time* and *space*) needed to solve computational problems. Meaningful analysis of the complexity of algorithms may only take place in the context of an agreed model of computation, or *machine model*. Many different machine models have been proposed in the past, including the Deterministic Turing Machine, Boolean circuit [6, 11] and P-RAM [8, 10]. The nascent field of DNA computing also suffers from the problem of proliferation of machine models. Several models have been proposed, within which we may construct algorithms for the solution of computational problems. However, complexity analyses of algorithms within different models of DNA computation are meaningless, since there are no uniform definitions of the concepts of time and space. Furthermore, if we are to compare a DNA-based model with a more traditional machine model, we require a way of demonstrating equivalence between the two.

In this paper we analyse the complexities of algorithms within a commonly employed model of DNA computation. We argue that this model, which we call the *weak model*, is actually inadequate from the point of view of obtaining *realistic* complexity results. This leads us to define a new *strong model* within which we reassess some claims that have been made concerning complexity of computations.

The paper is organised as follows. In Section 2 we recall the *weak model* of DNA computation first explicitly described in [4] although not so named there. We explain the shortcomings of this model and introduce the *strong model*, allowing us to make meaningful comparisons between DNA-based and more traditional models of DNA computation. In Section 3 we compare time complexities of extant algorithms within both the weak and strong models. In Section 4 we discuss the complexity of one extant Turing-complete model of DNA computation in the context of the strong model. In Section 5 we review the current search for the "killer application"; the one application that will establish a niche for DNA-based models of computation. We argue that the basis for such a quest is flawed, and suggest a potentially more fruitful line of enquiry in the light of the strong model.

2 Weak and strong models

Attempts have been made to characterise DNA computations using traditional measures of complexity, such as time and space. Such attempts, however, are misleading due to the nature of the laboratory implementation of the computation. We first examine these algorithms from a time complexity standpoint. Most extant models quantify the time complexity of DNA-based algorithms by counting the number of "biological steps" required to solve the given problem. Such steps include the creation of an initial library of strands, separation of subsets of strands, sorting strands on length, chopping and ligating strands.

Within these models, operations such as those described above are considered to be atomic actions performed in constant time. This assumption is patently false. In this section we rigorously define the time complexity of various laboratory operations, so that an accurate assessment of various DNA-based algorithms may be made.

2.1 The weak model

Here we recall [4] the basic legal operations on sets within what we now refer to as the *weak* model. The operation set described here is constrained by biological feasibility, but all operations are currently realisable with current technology.

- $remove(U, \{S_i\})$. This operation removes from the tube U, in parallel, any string which contains at least one occurrence of any of the substrings S_i.
- $union(\{U_i\}, U)$. This operation, in parallel, creates the tube U which is the set union of the tubes U_i.
- $copy(U, \{U_i\})$. In parallel, this operation produces a number of copies, U_i, of the tube U.

- *select*(U). This operation selects an element of U uniformly at random, if U is the empty set then *empty* is returned.

From the point of view of establishing the parallel time complexities of algorithms within the model, these basic operations are assumed to take constant time. This assumption has been commonly made by many authors in the literature [4, 13, 14]. However, these operations are frequently implemented in such a way that it is difficult to sustain this claim. For example, the *union* operation consists of pouring a number of tubes into a single tube, and this number is usually, in some way, problem size dependent. Assuming that in general we have a single laboratory assistant, this implies that such operations run in time proportional to the problem size.

Obviously, in the general case, a single laboratory assistant may not pour n tubes into one tube in parallel, nor may s/he split the contents of one tube into n tubes in parallel. This observation, if we are to be realistic in measuring the complexity of DNA computations, requires us to introduce the following constant time atomic operation:

- *pour*(U, U'). This operation creates a new tube, U, which is the set union of the tubes U and U'.

As we have observed, the *pour* operation is a fundamental component of all compund operations. It therefore follows that more realistic analyses of the time complexities of algorithms may be obtained by taking this operation into consideration.

2.2 The strong model

In what follows we refine the weak model just described. We assume that the initial tube (which takes at most linear time to construct) is already set up.

The *pour* operation is fundamental to all compound operations within our weak model. We must therefore reassess the time complexity of these operations. The *remove* operation requires the addition to U of

1. i tubes containing primers, and
2. A tube containing restriction enzymes

This operation is inherently sequential, since there must be a pause between steps 1 and 2 in order to allow the primers to anneal correctly. Therefore, the *remove* operation takes $O(i)$ time. Creating the *union* of i tubes is an inherently sequential operation, since the technician must first pour U_1 into U, then U_2, and so on, up to U_i. Rather than taking constant time, the *union* operation actually takes $O(i)$ time. It is clear that the *copy* operation may be thought of as a reverse-*union* operation, since the contents of a single tube U are split into many tubes, $\{U_i\}$. Therefore, *copy* takes $O(i)$ time.

3 Complexity comparisons in the weak and strong models

In this section we compare time complexities for algorithms previously described [4] within both the weak and strong models. In particular, we examine in detail the problem of generating a set of permutations. This will characterise the general form of comparisons that can be made, so that in the space available we merely tabulate comparisons for other algorithms.

- **Problem: Permutations**

  Generate the set P_n of all permutations of the integers $\{1, 2, \ldots, n\}$.

- **Solution**

  - *Input*: The input set U consists of all strings of the form $p_1 i_1 p_2 i_2 \ldots p_n i_n$ where, for all j, p_j uniquely encodes "positio n j" and each i_j is in $\{1, 2, \ldots, n\}$. Thus each string consists of n integers with (possibly) many occurences of the same integer.
  - *Algorithm*

```
for $j = 1$ to $n$ do
    begin
    copy($U, \{U_1, U_2, \ldots, U_n\}$)
    for $i$=1, 2, ... , $n$ and all $k > j$
            remove($U_i, \{p_j \neg i, p_k i\}$)
    union($\{U_1, U_2, \ldots, U_n\}, U$)
    end
$P_n \leftarrow U$
```

  - *Complexity*: $O(n^2)$ parallel-time.

In [4] the authors claimed a time complexity for this algorithm of $O(n)$. We justify the new time complexity of $O(n^2)$ as follows: at each iteration of the **for** loop we perform one *copy* operation, n *remove* operations and one *union* operation. The *remove* operation is itself a compound operation, consisting of $2n$ *pour* operations. The *copy* and *union* operations consist of n *pour* operations.

Similar considerations cause us to reassess the complexities of the algorithms described in [4], according to the following table:

Algorithm	*Weak*	*Strong*
Three colouring	$O(n)$	$O(n^2)$
Hamiltonian path	$O(1)$	$O(n)$
Subgraph isomorphism	$O(n)$	$O(n^2)$
Maximum clique	$O(n)$	$O(n^2)$
Maximum independent set	$O(n)$	$O(n^2)$

Table 1: Time comparison of algorithms within the Weak and Strong models

Although we have concentrated here on adjusting time complexities of algorithms described in [4], similar adjustments can be made to other work. Examples are given in the following section.

4 Fully-algorithmic DNA computations

Several authors [5, 14, 16] have described models of DNA computation which are Turing-complete. In other words, they have shown that any process that could naturally be described as an algorithm can be realised by a DNA computation. Papers [5] and [16] essentially show how any Turing Machine computation may be simulated by the addition of a *splice* operation to the models already described in this paper. In [14], Ogihara and Ray describe the simulation of Boolean circuits within a model of DNA computation. The complexity of these simulations is therefore of general interest. Here space permits us only to examine one such simulation. We choose that of Ogihara and Ray [14].

The authors of [14] claim real-time simulation of the class NC^1 [15] in time proportional to the depth of the circuit. Recall that NC^1 defines the class of problems of size n solved by bounded fan-in circuits of $O(\log n)$ depth and polynomial size. We point out that, with a single laboratory assistant, this estimate of the time complexity should be proportional to the *size* of the circuit. Thus, the claim of polylogarithmic running time (with, incidentally, a polynomial volume of DNA) in the weak model translates to polynomial running time in the strong (realistic) model. Essentially, the simulation does not harness the massive potential for parallelism that DNA offers. We now justify this claim.

The simulation proceeds as follows. An n-input, m-output Boolean network is modelled as a directed acyclic graph, $S(V, E)$, in which the set of vertices V is formed from two disjoint sets: X_n, the *inputs* of the network (of which there are exactly n); and G, the *gates* (of which exactly m are distinguished as output gates). Each input vertex has in-degree 0 and is associated with a single Boolean variable, x_i. Each gate has in-degree 2 and is associated with some Boolean operation θ from a set of *basis* operations Ω. The m distinguished output gates - $t_1, t_2, \ldots, t_m$ - are conventionally regarded as having out-degree equal to 0. An assignment of Boolean variables from $< 0, 1 >^n$ to the inputs X_n ultimately induces Boolean values at the output gates $< t_1, \ldots, t_m >$. An n-input, m-output Boolean network, S, is said to compute an n-input, m-output

Boolean function,

$f(\mathbf{X_n}) :< 0,1 >^n \rightarrow < 0,1 >^m =_{def} < f^{(i)}(\mathbf{X_n}) :< 0,1 >^n \rightarrow \{0,1\} : 1 \leq i \leq m >$ if $\forall \alpha \in < 0,1 >^n \ \forall 1 \leq i \leq m \ t_i(\alpha) = f^{(i)}(\alpha)$.

The two standard complexity measures for Boolean networks are *size* and *depth*: the size of a network, S, denoted $\mathbf{C}(S)$, is the number of gates in S; its depth, denoted by $\mathbf{D}(S)$, is the number of gates in the *longest* directed path connecting an input vertex to an output gate.

For each i, $1 \leq i \leq m$, a pattern $\sigma[i]$ of DNA is fixed. The presence of $\sigma[i]$ signifies that G_i evaluates to 1. We now describe how the gates G are simulated. We begin with a description of the simulation of the gates at level 0 (i.e., the creation of an initial tube, U, containing strands representing the value and form of the inputs X_n). For each input X_i do the following:

- If the node computes the positive form of some x_j, *pour*$(U, \sigma[i])$ if x_j=1.
- If the node computes the negative form of some x_j, *pour*$(U, \sigma[i])$ if x_j=0.

Ogihara and Ray claim that "this requires only one step". Given the inherent sequentiality of the *pour* operation, this statement is clearly false. An n-input Boolean circuit with $O(1)$ set-up time requires n technicians. We now consider the simulation of gates at level $l > 0$. We omit detailed discussion of the implementation, and concentrate purely on the number of *pour* operations required at each level. Let $i_1, i_2, \ldots, i_a$ be the indices of gates at level $l-1$, and $j_1, j_2, \ldots j_b$ those of the gates at level l. After amplifying the contents of the working tube, for each s, $1 \leq s \leq b$ the operation *pour*$(U, \sigma[j_s])$ is executed. This takes s time. Now U contains many copies of the strand representing each gate at level l. In order to simulate the operation of some gate G_s, two "linker" strands, representing the inputs to G_s are poured. This takes $2s$ time. Ogihara and Ray claim that they only require $\mathbf{D}(S)$ ligation steps during the course of the simulation. However, we believe that *in the general case* it is more meaningful to talk in terms of $O(\mathbf{C}(S))$ *pour* operations, even if we discount set-up time.

Despite these remarks, Ogihara and Ray's work is important because it establishes the Turing-completeness of DNA computation. This follows from the work of Fischer and Pippenger [7] and Schnorr [17], who described simulations of Turing Machines by combinational networks. Although a Turing Machine simulation using DNA has previously been described by Reif [16], Ogihara and Ray's method is simpler, if less direct.

5 Conclusions

In this paper we have emphasised the rôle that complexity considerations are likely to play in the identification of "killer applications" for DNA computation.

We have examined how time complexities have been estimated currently within the literature. We have shown that these are often likely to be inadequate from a realistic point of view. In particular, many authors implicitly assume that arbitrarily large numbers of laboratory assistants are available for the mechanical handling of tubes of DNA. This has often led to serious under-estimates of the resources required to complete a computation.

We have proposed a so-called *strong* model of DNA computation, which we believe allows *realistic* assessment of the time complexities of algorithms within it. This model, if the *splice* operation is trivially included, not only provides realistic estimates of time complexities, but is also Turing-complete.

We believe that success in the search for "killer applications" is the only means by which there will be sustained interest in DNA computation. Success is only a likely outcome if DNA computations can be described which will require computational resources of similar magnitude to those required by conventional solutions. At present, we believe that no realistic estimates of time complexities of DNA computations have been made, despite the claims of some authors. However, if, for example, we were to establish polylogarithmic time computations using only a polynomial volume of DNA, then this would be one scenario in which "killer applications" might well ensue. In this case, we might imagine that the vast potential for parallelisation would yield feasible solutions to very much larger problem sizes than could be achieved using existing, silicon-based parallel machines.

We hope that this paper will stimulate and focus interest in what we believe are very important issues for the future of DNA computation.

References

[1] Leonard Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley, 1974.

[3] American Mathematical Society. *Proceedings of the Second Annual Meeting on DNA Based Computers*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science., June 1996. To appear.

[4] Martyn Amos, Alan Gibbons, and David Hodgson. Error-resistant implementation of DNA computations. In *Proceedings of the Second Annual Meeting on DNA Based Computers* [3]. To appear.

[5] Erzsébet Csuhaj-Varjú, R. Freund, Lila Kari, and Gheorghe Păun. DNA computation based on splicing: universality results. In Lawrence Hunter and Teri Klein, editors, *Biocomputing: Proceedings of the 1996 Pacific Symposium.* World Scientific Publishing Co., Singapore, January 1996.

[6] Paul E. Dunne. *The Complexity of Boolean Networks.* Academic Press, 1988.

[7] M. Fischer and N.J. Pippenger. Relations among complexity measures. *Journal of the ACM*, 26:361–381, 1979.

[8] Steven Fortune and James Wyllie. Parallelism in random access machines. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing*, pages 114–118, San Diego, California, 1978.

[9] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[10] A. Gibbons and W. Rytter. *Efficient Parallel Algorithms.* Cambridge University Press, 1988.

[11] M.A. Harrison. *Introduction to switching and automata theory.* McGraw-Hill, 1965.

[12] Juris Hartmanis. On the weight of computations. *Bulletin of the European Association For Theoretical Computer Science*, 55:136–138, 1995.

[13] Richard J. Lipton. DNA solution of hard computational problems. *Science*, 268:542–545, 1995.

[14] Mitsunori Ogihara and Animesh Ray. Simulating Boolean circuits on a DNA computer. Technical Report 631, University of Rochester, August 1996.

[15] N. Pippenger. On simultaneous resource bounds. In *20th Annual Symposium on Foundations of Computer Science*, pages 307–311, Long Beach, Ca., USA, October 1979. IEEE Computer Society Press.

[16] John H. Reif. Parallel molecular computation: Models and simulations. In *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, Santa Barbara, June 1995.

[17] C.P. Schnorr. The network complexity and Turing machine complexity of finite functions. *Acta Informatica*, 7:95–107, 1976.