Relating the Minimum Model for DNA Computation and Boolean Circuits

Mitsunori Ogihara*
Department of Computer Science
University of Rochester
Rochester, NY 14627-0226

April 29, 1999

Abstract

This paper provides a refinement of the minimum DNA computational model by [OR98b], which assumes the smallest set of permissible biochemical operations. The power of the refined model is characterized in terms of standard Boolean circuit complexity classes.

1 Introduction

Following seminal work by Adleman [Adl94], the potential of DNA as an alternative device for massively parallel computation has been actively studied (for a survey see [ORS97]). One important issue in this field is to clarify how much power DNA based computation can have. A number of papers have studied this problem [Bea95,BDLS96,FB97,Hag98,OR98b,Rei95,RW96,Win96], but the results are quite diverse. This is partly because DNA computation can be formulated in various ways. One can think of computation as working on a huge collection of library strands that have been generated prior to actual computation, as repeating generation processes so that all members are eventually produced, as processing a small collection of short fragments of DNA, and so on. The diversity is also due to the fact that it is not clear what operations are admissible for DNA computation. Current technical limitations may be overcome in the future.

The focus of the present paper is DNA computation carried out on a small set of short DNA patterns. We maintain a subset of that pattern set, which is realized as a test tube. Each computation phase consists of a number of biochemical operations with which we generate another collection of patterns. After applying such phases a number of times we come to the final step of computation in which we test whether the final test tube contains a strand at all. That test result is translated into a single bit output, which is the outcome of computation. Such style of computation is studied in [OR97,OR98b,ADG98]. The paper [OR98b] studies such kind of computation with the smallest set of operations and with a reasonable asymptotic increase in the amount of DNA required (more precisely, a polynomially bounded volume requirement). It is shown in that paper that the power of such computation resides between NC, the polynomial-size, bounded-fan-in circuits, and SAC, the polynomial-size, semi-unbounded-fan-in circuits. It is also shown that such DNA computation can be "robust" against uncertainty involved during computation by the use of an enzyme that decomposes single-stranded parts of partially-double strands.

The goal of this paper is to fill the gap between NC and SAC of the result shown in [OR98b]. We refine the minimum model and obtain complexity characterizing NC, SAC, and AC, the unbounded-fan-in-

*Supported in part by the National Science Foundation Grants CCR-9701911 and CCR-9725021.

circuits, in terms of DNA computation under that model. The equivalences suggest that DNA computation by fragments is essentially Boolean circuit computation.

2 Boolean Circuits

A *Boolean circuit* of n inputs is a directed acyclic graph with labeled nodes. There are exactly $2n$ nodes with indegree 0. These nodes are called input gates and are labeled $x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}$. Other nodes are labeled by one of `AND` and `OR`. A node labeled `AND` computes the conjunction of its input signals while one labeled `OR` computes the disjunction of its input signals. These gates are stratified and each level consists of the same kind of gates; that is, there are levels of `OR`-gates and those of `AND`-gates. There is a unique node with outdegree 0. The gate is called the output gate. The models of Boolean circuits are defined in terms of the gate fan-ins (the number of maximum inputs that `AND`-gates and `OR`-gates can take). In *bounded-fan-in circuits*, both `AND`-gates and `OR`-gates may have at most two inputs. In *unbounded-fan-in circuits*, both `AND`-gates and `OR`-gates may have an arbitrary number of inputs. In *semi-unbounded-fan-in circuits*, `OR`-gates can have an arbitrary number of inputs while `AND`-gates have at most two inputs. We envision that a Boolean circuit has alternating levels of `AND`s and `OR`s.

We are concerned with the following complexity measures for Boolean circuits. The *size* of a circuit C is the number of gates in it. The *depth* of C is the length of the paths from input gates to the output gate. The *fan-out* is the maximum number of outgoing edges of any gate of C. The *AND-fan-in* is the maximum number of incoming edges of any `AND` gate in C. The *OR-fan-in* is the maximum number of incoming edges of any `OR` gate in C. A *language* is a collection of nonempty words over $\{0,1\}$. A language L is *decided* by a family $\{C_n\}_{n\geq 1}$ of Boolean circuits if for every $n \geq 1$ C_n takes n bit string as an input and outputs 1 only on words in L of length n. For a function $D(n)$, $\mathrm{NC}(D(n))$ (respectively, $\mathrm{SAC}(D(n))$ and $\mathrm{AC}(D(n))$) denotes the class of languages recognized by a family of bounded-fan-in (respectively, semi-unbounded-fan-in and unbounded-fan-in) circuits of polynomial size, depth $D(n)$.

3 The Refined Minimum Model

The permissible operations of the minimum DNA computation model [OR98b] are: merge (mixing the contents of two test tubes into one), synthesize (chemically synthesize specific patterns of DNA), anneal (make DNA strands form double strands without elongation), denature (the reverse of anneal), detect (test whether a test tube contains a DNA strand or not), and length (separate DNA strands according to their lengths).[1] These are quite well-studied biochemical operations [SFM89]. We note that recent technical advances have made the length technique robust and rapid [ROMJ97]. Why does the model need to have these operations? The necessity of the first four operations is clear. As to the last operation, DNA computation cannot be carried out unless there is a method for taking out some strands out of a test tube. There are two such operations, i.e., sequence-specific separation and length-specific separation, the length operation. In comparison, the length-specific one looks less powerful than the other.

The uncertainty involved in these operations is represented by the parameter ϵ, $0 < \epsilon < 1$. We make the following assumptions on uncertainty:[2]

1. In the merge operation as well as in the length operation at most an ϵ fraction of the contents can be lost.

[1] In [OR98b] the denaturing operation is a part of the length operation.

[2] [OR98b] uses distinct parameters for these events.

2. In the anneal operation at most an ϵ fraction of the strands may remain unhybridized even though their mates exist in the test tube or may form incorrect hybridization due to mismatch.

3. The yield of the synthesis operation is controllable only within the factor of $(1+\epsilon)$: the actual yield of synthesis that is configured for M molecules is in the range of $[M/(1+\epsilon), M \cdot (1+\epsilon)]$.

The second kind of uncertainty is resolved by the use of Mung Bean nuclease [MHDM84]. Mung bean nuclease breaks strands with incorrect hybridization into completely-double-stranded pieces, thereby making them shorter. If the strands with everywhere correct hybridization are completely double and are the longest of all possible compounds generated by anneal, then the use of the enzyme will allow us to discriminate correct ones against the rest. We will refer to the use of Mung Bean nuclease for elimination of single DNA strands by *degenerate*.

Note that with the minimum set of operations, computation under the model should be broken into cycles of six sequential steps:

$$\text{merge} \rightarrow \text{anneal} \rightarrow \text{degenerate} \rightarrow$$
$$\text{length} \rightarrow \text{denature} \rightarrow \text{length}.$$

The block [anneal $\rightarrow$ degenerate $\rightarrow$ length] is for "composing" patterns of DNA and the block [denature $\rightarrow$ length] is for "selecting" necessary strands. A *DNA program* under the minimum model is a sequence of biochemical operations. A program is split into two parts: *preparation* and *experiment*. The preparation consists of synthesis and merge. It produces a number of test tubes that contain DNA strands necessary for computation. These test tubes are considered as the *hardware*. The experiment does the rest, namely execution of the above computation cycles. The last operation in the experiment part is always detect, when the outcome—whether there are strands in the test tube or not—is translated into the output bit of the program. If the program is for length n inputs there are $2n$ special ones in the "hardware" test tube, labeled by $x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}$. Prior to the experiment part, for each i, $1 \leq i \leq n$, we select x_i if the i-th input bit is a 1 and $\overline{x}_i$ otherwise, then we mix the n tubes together. This is the generation of the *input test tube*. A DNA program for length n inputs *robustly* decides a collection A of length n words if for every word w of length n, the following two properties hold:

- If w belongs to A then the program on "input" w outputs 1 regardless of the uncertainty involved. To be more precise, so long as the uncertainty occurs within the factor of ϵ, every time we run the program on w, we obtain 1 as the outcome.

- if w does not belong to A then the program on "input" w outputs 0 regardless of the uncertainty involved.

A family $\{P_n\}_{n\geq 1}$ of DNA programs *decides* a language L if for every $n \geq 1$, P_n robustly decides the length n part of L.

Complexity Measures

As in [OR97,OR98b,OR98a,ADG98] we view our computation model as some kind of circuit, where signals are single stranded DNA produced at the end of the cycles and the gates are the double strands that are formed by the anneal operation. To draw a more precise relation, we introduce a number of complexity measures

The *duration* is the number of operational steps in the experiment part of the program. The *size* is the total number of single or completely double strands that appear during at any time of computation. The *distribution factor* is the maximum number of compounds that share the same pattern of single DNA strand as a component. The *decomposition factor* is the maximum number of distinct (in term of patterns) single DNA strands that form any legitimate double compound. Finally the *word size* is used to measure how

long a compound can be. We assume that each compound can be split up into short stretches of DNA of equal length. Those short stretches of DNA are called the *base strands*. We measure the length of a compound by the ratio of its length over the length of base strands. The maximum of these ratios for all legitimate compounds is the word size.[3] We note that the length of the base strands can be proportional to the logarithm of the number of the base strands [Bau99]. We assume that no legitimate single DNA is a substring of another; this ensures that there is always a unique way of composing each legitimate compound.

The following is essentially proven in [OR98b].

Theorem 1 *1. There exists a constant $c > 0$ such that for every $S(n)$ and $D(n) = \Omega(\log n)$, every language decided by a family of bounded-fan-in circuits of size $S(n)$ and depth $D(n)$ is robustly decided by a family of DNA programs of size $cS(n)$, duration $cD(n)$, word size 2, distribution factor 2, and decomposition factor 2.*

2. There exists a constant $c > 0$ such that for every $S(n)$ and $D(n)$, every language robustly decided by a family of DNA programs of size $S(n)$, duration $D(n)$, word size 2 is decided by a family of semi-unbounded-fan-in circuits of size $cS(n)D(n)$ and depth $cD(n)$.

4 New Results

We relate the minimum model and the circuit model. We show that there are close relationships between the size of a DNA program and that of a circuit, between the distribution factor and both the fan-out and the OR-fan-in, between the decomposition factor and the AND-fan-in, and between the duration and the depth.

For functions $S(n), D(n), M(n), F(n), L(n)$ from the set of natural numbers to itself, let

$$\mathrm{MM}(S(n), D(n), M(n), F(n), L(n))$$

denote the class of languages robustly decided by a family of programs under the minimum model of size bounded by $S(n)$, duration bounded by $D(n)$, distribution factor bounded by $M(n)$, decomposition factor bounded by $A(n)$, and word size bounded by $L(n)$. For functions $S(n), D(n), M(n), F(n), G(n)$ from the set of natural numbers to itself, let

$$\mathrm{CC}(S(n), D(n), M(n), F(n), G(n))$$

to denote the class of languages robustly decided by a family of programs under the minimum model of size bounded by $S(n)$, depth bounded by $D(n)$, fan-out bounded by $M(n)$, AND-fan-in bounded by $F(n)$, and OR-fan-in bounded by $G(n)$. Here we allow function classes to appear in place of functions. Let $poly$ and $const$ respectively denote the class of all polynomials and that of all constants. Note for every $D(n)$, that

$$\begin{aligned}
\mathrm{NC}(D(n)) &= \mathrm{CC}(poly, D(n), poly, const, const), \\
\mathrm{SAC}(D(n)) &= \mathrm{CC}(poly, D(n), poly, const, poly), \\
&\quad \text{and} \\
\mathrm{AC}(D(n)) &= \mathrm{CC}(poly, D(n), poly, poly, poly).
\end{aligned}$$

Theorem 2 *There exists a constant $c > 0$ such that for every $S(n), D(n), L(n), M(n)$, and $F(n)$, the following relations hold:*

1. $\mathrm{MM}(S(n), D(n), M(n), F(n), L(n))$
   $\subseteq \mathrm{CC}(cS(n)D(n), cD(n), M(n), F(n), M(n))$.

[3] In [OR98b] the word size is set to a constant.

2. $\mathrm{CC}(S(n), D(n), M(n), F(n), G(n))$
$\subseteq \mathrm{MM}(cS(n), cD(n), c\max\{M(n), F(n)\},$
$cF(n), c\max\{M(n), F(n)\})$.

Proof (sketch) For Property 1, let us fix input size n thereby fixing the program of our concern. Let us also fix an enumeration $w_1, \ldots, w_K$ of all the legitimate strands that can appear during computation (single or double). Let D denote the duration of the program. For each k, $1 \leq k \leq K$, and each d, $0 \leq d \leq D$, let $v(k, d)$ be the Boolean variable which expresses whether the k-th strand is present in the reaction test tube at time d. We create a Boolean circuit that computes all these variables concurrently for all k, proceeding from $d = 1$ through $d = D$. We show that for each computational step d, the values of $v(k, d)$ for each k can be computed by a depth-2 circuit that takes inputs from $v(k', d')$ for $d' < d$ and has AND-fan-in bounded by $M(n)$ and OR-fan-in bounded by $F(n)$. We show this operation by operation.

As to merge, there are only two test tubes involved. Let d' and d'' be the time steps from which the two test tubes come. Then the circuit for $v(k, d)$ is $v(k, d') \vee v(k, d'')$ if both d' and d'' are greater than 0. If one of these is 0, the corresponding test tube is from the preparation step. Since we do know (now that the program has been fixed) the contents of the test tube, one of the two inputs can be replaced by a 1 or a 0 depending on whether the k-th compound is in it or not.

As to degenerate we can assume, due to the robustness of the program, all incorrect hybridized strands as well as all those with single stranded parts are eliminated. The values of them are simply set to 0 and the other are kept the same. As to length we keep the values of those with the specified length and set the others to 0.

As to anneal, for a double compound w_k, it is present after anneal if and only if all the components necessary for w_k are present, so $v(k, d)$ is computed as a conjunction of at most $F(n)$ variables. Due to uncertainty, for every single strand that is present in the test tube before annealing, it can happen that one copy of that strand will remain unhybridized during the annealing step. So the values of single strands should be kept the same (it will become 0 in the subsequent degenerate step, though).

As to detect, we look at a single variable, so the circuit is of depth 1.

As to denature, each double compound will have value 0 while a single compound has the value equal to the disjunction of all the double compounds in which it can appear. The number of such double compounds is bounded by $M(n)$.

Overall, computation of any variable can be done either using an `OR` of $M(n)$ things (assuming that $M(n) \geq 2$) or using an `AND` of $F(n)$ things.

The entire circuit has depth $2D(n)$ since we need to stratify the circuit. It has the total of $S(n)D(n)$ gates at most. The fan-out is clearly the distribution factor. This proves Property 1.

As to Property 2, we will extend the method in [OR97]. Each gate corresponds to a unique strand chosen from the base strands. For a gate g, let $\sigma[g]$ denote the strand corresponding to it.

In the case when g computes the `OR` of $h_1, \ldots, h_m$, we pour $\sigma[g]$ as well as the complementary antiparallel strands of $\sigma[g]\sigma[h_1], \ldots, \sigma[g]\sigma[h_m]$ into the test tube and attempt to create m double strands; i.e., the complete double strands with $\sigma[g] \cdot \sigma[h_1], \ldots, \sigma[g] \cdot \sigma[h_m]$ on one side, where $\sigma \cdot \tau$ represents σ followed by τ with the two strands disconnected. After degeneration we take out "double" length DNA without denaturing, which operation will fish out all the complete double strands we have synthesized. Then we denature and remove double length strands thereby all the "linker" strands.

In the case when g is an `AND`-gate we will append all of $\sigma[h_1], \ldots, \sigma[h_m]$ to $\sigma[g]$ in this order, which will create $(m + 1)$ times longer strands.

We need to worry about contention for the same single strand by many compounds. In order to avoid a situation in which a compound corresponding to a gate outputting 1 does not get all of necessary strands, we need to be careful about how many copies of $\sigma[g]$ and the linkers are poured. Based on the value of ϵ,

for each d such that the d-th operation is the end of a cycle, we compute two values LB_d and UB_d such that for each single strand present in the test tube, the number of its copies is at least LB_d and at most UB_d. The computation of these values can be done since we know the input Boolean circuit. The synthesis operation for the test tube contents necessary for the subsequent annealing step is done so that each merge strand has at most $LB_d/fan\text{-}out$ copies. Then there will be no contention.

Due to the uncertainty and due to fan-out, we produce less and less single strands. So, we need to amplify them from time to time. In order to amplify the presence of strand θ by a factor of Q (right after the end of a cycle, say the d-th step), we translate each θ into Q' copies of some strand θ'. For that matter we put at least $Q \cdot UB_i$ copies of θ' and at least UB_i copies of the strand complementary antiparallel to $\theta\underbrace{\theta' \cdots \theta'}_{Q'}$. If Q' is set large enough, even with the uncertainty involved, the amplitude of θ' will become at least Q times that of θ. The maximum amplification factor is, if we are going to do this each time a level has been simulated, will be bounded either by a constant times the fan-out or by a constant times the AND-fan-in, whichever the larger. This proves Property 2. ∎

Corollary 3 *For every $D(n)$,*

$$\begin{aligned}
&\mathrm{NC}(O(D(n))) \\
&\quad = \mathrm{MM}(poly, O(D(n)), const, const, const), \\
&\mathrm{SAC}(O(D(n))) \\
&\quad = \mathrm{MM}(poly, O(D(n)), poly, const, poly), \text{ and} \\
&\mathrm{AC}(O(D(n))) \\
&\quad = \mathrm{MM}(poly, O(D(n)), poly, poly, poly).
\end{aligned}$$

Proof (Sketch) The last two equalities are straightforward. As to the first, [HKP84] shows that each bounded-fan-in circuit can be modified to an equivalent bounded-fan-in circuit with fan-out 2 with only linear increase in the size and the depth. Thus $\mathrm{NC}(D(n)) = \mathrm{CC}(poly, D(n), poly, const, const)$. Combining this with Theorem 2 yields the equality. ∎

5 Conclusion

We showed tight relationships between the refined minimum model and the standard Boolean circuit model. It is an interesting question how the constant factors involved in the simulations can be reduced to tighten the relationships.

References

[ADG98] M. Amos, P. E. Dunne, and A. Gibbons. DNA simulation of boolean circuits. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. D. Deb, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. L. Riolo, editors, *Proceedings of 3rd Annual Genetic Programming Conference*, pages 679–683, San Francisco, CA, 1998. Morgan Kaufmann.

[Adl94] L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.

[Bau99] E. Baum. DNA sequences useful for computation. In L. Landweber and E. Baum, editors, *DNA Based Computers II*, pages 235–242. The American Mathematical Society DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume **44**, 1999.

[BDLS96] D. Boneh, C. Dunworth, R. Lipton, and J. Sgall. On the computational power of DNA. *Discrete Applied Mathematics*, 71:79–94, 1996.

[Bea95] D. Beaver. Computing with DNA. *Journal of Computational Biology*, 2(1):1–8, 1995.

[FB97] B. Fu and R. Beigel. A comparison of resource-bounded molecular computation models. In *Proceedings of the 5th Israel Symposium on Theory of Computing and Systems*, pages 6–11. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[Hag98] M. Hagiya. Towards autonomous molecular computers. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. D. Deb, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. L. Riolo, editors, *Proceedings of 3rd Annual Genetic Programming Conference*, pages 691–699, San Francisco, CA, 1998. Morgan Kaufmann.

[HKP84] H. J. Hoover, M. M. Klawe, and N. J. Pippenger. Bounding fan-out in logical networks. *Journal of the Association for Computing Machinery*, 31(1):13–18, 1984.

[MHDM84] T. F. McCutchen, J. L. Hausen, J. B. Dame, and J. A. Mullino. Mung bean nuclease cleavage *Plasmodium* genomic DNA at site, before and after genes. *Science*, 225:626–628, 1984.

[OR97] M. Ogihara and A. Ray. Simulating boolean circuits on DNA computers. In *Proceedings of 1st International Conference on Computational Molecular Biology*, pages 326–331. ACM Press, 1997. *Algorithmica*, to appear.

[OR98a] M. Ogihara and A. Ray. A DNA-based self-propagating algorithm for solving bounded-fan-in Boolean circuits. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. D. Deb, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 725–730, San Francisco, CA, July 1998. Morgan Kaufman.

[OR98b] M. Ogihara and A. Ray. The minimum DNA model and its computational power. In *Unconventional Models of Computation*, pages 309–322. Springer, Singapore, 1998.

[ORS97] M. Ogihara, A. Ray, and K. Smith. Biomolecular computing—a shape of computation to come. *SIGACT News*, 28(3):2–11, 1997.

[Rei95] J. Reif. Parallel molecular computation. In *Proceedings of 7th ACM Symposium on Parallel Algorithms and Architecture*, pages 213–223. ACM Press, 1995.

[ROMJ97] B. B. Rosenbaum, F. Oaks, S. Menchen, and B. Johnson. Improved single-stranded DNA sizing accuracy in capillary electrophoresis. *Nucleic Acids Research*, 25:3925–3929, 1997.

[RW96] D. Rooß and K. Wagner. On the power of DNA computing. *Information and Computation*, 131:95–109, 1996.

[SFM89] J. Sambrook, E. F. Fritsch, and T. Maniatis. *Molecular Cloning: a Laboratory Manual*. Cold Spring Harbor Press, NY, 2nd edition, 1989.

[Win96] E. Winfree. Complexity of restricted and unrestricted models of molecular computation. In R. Lipton and E. Baum, editors, *DNA Based Computers*, pages 187–198. The American Mathematical Society DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume **27**, 1996.