Recent Developments in DNA-Computing

Diana Rooß
Lehrstuhl für Theoretische Informatik
Universität Würzburg
Am Exerzierplatz 3, 97072 Würzburg, Germany
diana@informatik.uni-wuerzburg.de

Abstract

In 1994 Adleman published the description of a lab experiment, where he computed an instance of the Hamiltonian path problem with DNA in test tubes. He initiated a flood of further research on computing with molecular means in theoretical computer science. A great number of models was introduced and examined, concerning their computional power (universality as well as time and space complexity), their efficiency and their error resistance. The main results are presented in this survey.

1 Introduction

The pioneering achievement that every problem can be computed, providing that one can verbalize a method to solve it, belongs to the second third of our century. Simultanously the answer to the question "how to do it" was found. The necessary information was encoded binary, and the code words were implemeted by a sequence of high and low voltage. This was the beginning of the phenomenal expansion of computers. A lot of innovations supported its breakthrough. But the idea to store and modify data by electronic means has not changed in practice.

The description of a biological experiment caused a sensation among theoretical computer scientists in December 1994. A well-known scientist succeeded in computing the solution of an instance of a problem recognized to be hard, by means of biochemical manipulation of DNA (desoxyribonuclein acid): Adleman solved the Hamiltonian path problem for a graph with seven vertices using a soup of DNA.

Though in this computation the means of storage was new the result was not. For a long time it has been possible to search for Hamiltonian paths in graphs of this size, and the usual method in this case is much faster than the new one by Adleman. However, the complexity theoretical analysis of this experiment indicates that the input length and the computing time to solve the task are in good proportion. Adleman's procedure uses polynomial computing time, but so far there are only exponential time algorithms known to compute the Hamiltonian path problem on a sequential machine, and this does not rank as efficiently computable.

A Hamiltonian path is a sequence of edges in a graph, which touches every vertice exactly once. The Hamiltonian path problem is to decide, whether a graph has a Hamiltonian path or not. This problem is a typical representative of the important complexity class NP. That means that it shares many of its properties, including those mentioned above, with every problem in NP. Adlemans method has initiated great interest, as one can hope to learn something new about all of the problems in NP by analysing this special case.

Massively parallel features are the reason for the gain of time (considering larger inputs). The solution space is completely coded, each possible solution is mapped to a DNA molecule. Subsequently some necessary conditions for a correct solution are checked step by step. In each of the steps all molecules are modified in parallel: those molecules that fit to the tested condition are seperated from those which do not.

A possible solution to the Hamiltonian path problem is a sequence of the graph's vertices. There are exponentially many sequences, but each of them has only polynomial length and therefore it differs from other sequences at most at polynomially many positions. Consequently at most polynomially many conditions have to be tested. As all possible solutions are tested in parallel, their huge number is of no consequence when counting the time.

If we generalize Adleman's method we get a parallel model of computation. We view molecules as distributed data storage and molecular manipulations as parallel operations. In a test tube a soup of millions of molecules can be manipulated whereas silicon-based parallel computers are bounded to about ten thousand processors nowadays.

Such considerations encourage to ask, whether DNA-computers can break the "exponential border" and therefore problems , that are not computable in a practical sense nowadays (as this would take thousands of years), could be solved in a sensible time. But simple means of calculation shows, that Adleman's procedure would need more molecules than the earth is able to give, for computing the Hamiltonian path problem for a graph with two hundred vertices [23, 34].

It is the task of theoretical computer science to find out the chances and the borders of the new computational model. The survey in hand first presents the article that initiated research on molecular computing. We consider the description of the experiment as well as Adleman's hopes and doubts concerning this procedure (Section 2). In Section 3 we explain the first abstractions of Adleman's method towards a parallel computation model and its complexity theoretic analysis. Some more molecular operations and tests are mentioned which are expected to be realizable. "Is everything computable by DNA-computers?" We trace this question in the Section 4. In fact there are different DNA models that have the power of an universal computer. There are many problems which can only be solved with the machines of the preceeding sections taking great consumption of time and DNA resourses into the bargain. The efficient algorithms of the fifth section for example reduce the consumption of molecules considerably, compared with naive methods. In Section 6 we face the objection that molecular operations are probabilistic – so error analysis and strategies for error resistence are necessary. Finally (Section 7) we present a research area from the theory of formal languages: splicing systems model a biological prototype where DNA molecules are reorganized by the means of enzymes and ligation. Many classes of languages can be characterized by different modifications of splicing systems.

Adleman's article caused a flood of further research work in various aspects and developments of molecular models. For that reason it is impossible to give a complete state of research in this presentation. Instead we select some papers which seem to be interesting to us and which afford an insight into the ways research on molecular computing has taken. For clearing the basics of theoretical computer science we recommend for example [49].

2 Adleman's Experiment

In a well-noticed article from December 1994 Adleman describes a completely new process for solving a combinatorial problem that he performed for an examplary input in lab [1]. He takes the NP-complete Hamiltonian path problem as representative of an important complexity class which is supposed to have exponential worst case time complexity.

Let G be a graph with n vertices, where vertices v_{in} and v_{out} are marked. G is called to have a *Hamiltonian path from v_{in} to v_{out}* if there is a path of edges starting with v_{in} and ending with v_{out} that contains every vertice of G exactly once. The *directed Hamiltonian path problem* is the set of tuples (G, v_{in}, v_{out}) where G has a Hamiltonian path from v_{in} to v_{out}.

Adleman uses the following (nondeterministic) algorithm to solve the directed Hamiltonian path problem for an input (G, v_{in}, v_{out}).

1. Generate random paths in G.
2. Extract all paths beginning with v_{in} and ending with v_{out}.
3. Extract all paths with length exactly $n - 1$.
4. Extract all paths that contain every vertice at most once.
5. Accept if there are any paths left, otherwise reject.

Those steps are now realized as molecular computation phases. Vertices and edges of G are coded by DNA polymers. Ligation builds DNA strands that represent random paths in G (step 1). The Watson-Crick complements of the codings of v_{in} and v_{out} are used to extract the strands with the correct beginning and end (step 2). In order to get codings of length $n - 1$ we seperate the strands in agarose gel (step 3). Next we denaturate the DNA, and we check each vertice (using the Watson-Crick complement of its coding) for only single appearance in a path (step 4). To get the result, again we use gel electrophorese for testing whether there is any strand left or not (step 5). In between the steps polymere chain reaction (PCR) is used to amplify the intermediate results.

The execution of the experiment took seven days of lab work. But Adleman remarks that optimizing the algorithms and the molecular implementation of the operations can reduce the time substantially. However, the time used for longer inputs grows only linearly with the number of vertices. On the other hand the number of different paths grows exponentially with the number of vertices – and so does the amount of DNA. Adleman himself poses some questions. He consideres that the proceeding has probabilistic properties. That means it has to be examined how many *equal* strands must be initialized, to reach high probability for the existence of at least one of the required coding in every step. All together careful error analysis and the examination of the realizability of the method is necessary [2, 28].

For all that Adleman rates the potential of his DNA model to be promissing. While modern supercomputers perform 10^{12} operations per second, he estimates 10^{20} operations per second to be realistic for molecular manipulations. Similar impressive views concern the consumption of energy and the capacity of memory: supercomputers need one joule for 10^9 operations, whereas the same energy is sufficient to perform $2 \cdot 10^{19}$ ligation operations. On a videotape every bit needs 10^{12} cubic nanometres storage; DNA stores information with a density of one bit per cubic nanometre (also see [6]).

It depends on future research whether the prototype of a reliable bio-computer or even industrial mass production will be reality. However, Adleman got an avalanche underway.

3 First Reactions

The first who takes up Adleman's method is Lipton [30, 32]. He makes some idealized assumptions and gets a deterministic parallel computation model. The molecular operations *union*, *initialization*, *extraction*, and *amplification* and an *emptiness test* are applicable on DNA in test tubes. By that he builds an abstraction of the molecular computation in Adleman's experiment. The main results of those papers are a concrete DNA algorithm for the 3-SAT problem and the proof that any problem in NP can be solved in polynomial time using his model.

Inspired by Lipton's abstraction Rooß and Wagner pick up his model and expand it with other molecular operations and tests that are supposed to be realizable [43, 44]. For the exact definition of the operations refer to the original articles, because the computational power depends on a few properties. In detail we can categorize the operations as follows.

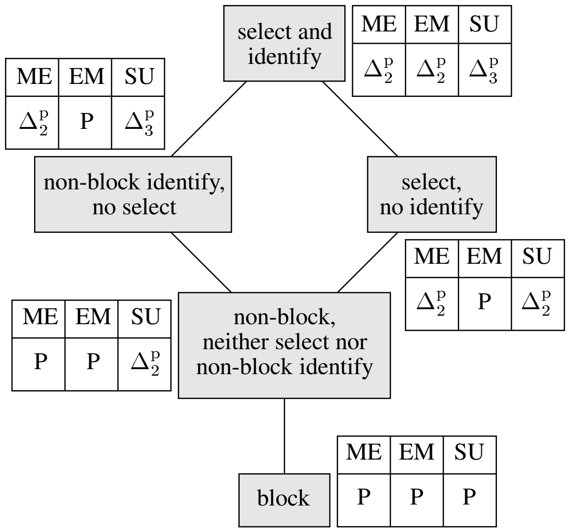
- Operations have the *block property* if they preserve the fact that all strands of a certain length are contained in a test tube or none of them;
- Operations have the *select property* if they can exclude certain strands from a test tube (and probably modify the remaining);
- Operations have the *identify property* if they can identify different strands (i. e. by identifying or cutting certain symbols or substrings).

We call the operations with the block property *block operations*, those with the select property *select operations* and those with the identify property *identify operations*. A *non-block operation* does not have block property, an operation with the identify and the block property is called *simple identify operation*.

Lipton uses the *emptiness test (EM)* that is true, if there are no strands in a test tube in question. Additionally we examine another two DNA tests: The *membership test (ME)* is true, if there is a certain strand in the test tube in question; the *subset test (SU)* is true, if the contents of a certain test tube is as a subset in the test tube in question.

To construct an algorithm for the model, on the one hand the set of commands of classical Pascal, on the other hand a set of molecular operations and tests are allowed. Some variants of this DNA-Pascal are inspected, they differ in the set of valid molecular operations and tests. If we bound the resulting programming languages to polynomial computing time, they define different problem classes that can be characterized by classical complexity classes. The pure Pascal commands do not influence the magnitude of those classes because polynomially time-bounded Pascal exactly characterizes the class P.

We can summarize the results of [44] in the following figure. The vertices of the graph are combinations of operation properties. A tabular is attached to each of the vertices. It shows the computational power of polynomially time-bounded DNA-Pascal using operations with the properties in the vertices and different tests.



If only block operations are used no molecular test can increase the power of polynomially time-bounded DNA-Pascal above P. Non-block operations without select property and at most simple identify operations lift polynomially time-bounded DNA-Pascal to the magnitude of $\Delta_2^p = P^{NP}$.

The select property seems to turn the scale for the power of EM: with select operations and EM (or SU) polynomially time-bounded DNA-Pascal is as powerful as Δ_2^p. (Lipton's model can be classified at this point, its power is exactly characterized by Δ_2^p.) Without select operation EM does not have an influence on the power of the programming language.

A non-block identify operation lifts the power to Δ_2^p if ME is allowed – with SU the power is Δ_3^p. We get the same effect if only simple identify operations but additionally select operations are valid.

There are other operations which raise the power of Lipton's model even to PSPACE. If the strands are logarithmically length-bounded then DNA-Pascal with Lipton's operations and test is exactly characterized by the class L [43].

4 Universal Computers

The term *universal computer* originates from recursion theory and describes a machine that is able to solve any computable problem. As an input it obtains a program which implements an algorithm for solving a computable problem, and a parameter for that program. There are many equal models which fit for a universal computer. Probably the *Turing machine* is the most important one. Each Turing computation is represented by a sequence of configurations. Each configuration encodes a single step in the computation. Successor configurations differ only at a few locally bounded positions. A Turing machine accepts an input, if there is a sequence of successor configurations that begins with the start configuration and ends with the accepting configuration. The commonly accepted *Thesis of Church* says that any intuitively computable problem is computable by a Turing machine. According to these standards every new computational model is going to be examined whether it is an universal computer or not.

Beaver describes a molecular universal computer in [10]: he simulates a Turing machine by coding all its configurations in DNA strands. During that he starts with pairs of successor configurations. Similar to the dominoes game (where one chain of dominoes must be put together) longer and longer parts of the computation grow (in parallel) by clever recombination of these strands. Finally it is to check, whether there is a strand with the coding of the start configuration in the beginning and the accepting configuration in the end. In [45, 51] similar ligation-based methods are proposed to simulate Turing machines. From [43] it follows that without time bounds Lipton's model can already process any computable problem.

5 DNA Algorithms

Adleman's and Lipton's naive algorithms for the Hamiltonian path and the 3-SAT problem, are valuable contributions to the analysis of the computational power of those machines. But it appears that only small inputs can be processed in reality. The problem is the extend of solution spaces of order 2^n, where n is the input length. Indeed we can not expect to reduce the solution space for such problems to polynomial size with optimized algorithms because this would imply the equality of P and NP [43], nevertheless the number of the actually computable instances grows considerably. Bach, Condon, Glaser, and Tanguay [5] present clever algorithms that use Adleman's operations and test. For example they reduce the DNA consumption to an order of 1.51^n while computing the NP-complete *Independent Set problem*. Ogihara utilizes the Monien-Speckenmeyer algorithm to construct a DNA algorithm for 3-SAT that increases the maximal input length from 70 (specified by Lipton) to 120 [35]. Nature has a method to comb huge solution spaces: repeated selection cycles on much smaller subspaces. Dynamical programming picks up this idea to reduce the number of required DNA strands [48, 8].

Moreover, there are several other algorithms for famous difficult problems. E. g. the #P-complete problem of permantent calculation in a matrix is solved in [50, 29] by a DNA algorithm.

Not only the decision of sets with DNA computation is of interest. If molecular models want to compete with silicon-based models, the computation of functions must be possible. Within that the addition plays an important part; a corresponding DNA algorithm is presented in [22]. Molecular matrix multiplication is the theme in [37] and a quite early paper by Beaver deals with a factorisation algorithm [9].

Boolean circuits are one of the profound studied parallel computation models. Their processors are gates that execute the boolean functions "not", "and" and "or". The complexity of those circuits are given by the *fan-in* and *fan-out* (the maximal number of inputs and outputs, resp. of a gate), the *size* (the number of gates) and the *depth* (the longest way through the circuit). Often the fan-in of and-gates is bounded to two whereas the fan-in of or-gates is arbitrary; such circuits are called *semi-unbounded*. The simulation of boolean circuits by DNA-computers is among others examined in [31, 12, 27]. Ogihara and Ray [36] specify a molecular algorithm that simulates semi-unbounded boolean circuits of depth t and size g in time $\mathrm{O}(t \log f)$ and DNA consumption $\mathrm{O}(gf)$, where f is the fan-out of a circuit. In particular this makes it possible to run the simulation of the important circuit class NC^1 in time linear to the depth of the circuits.

DNA molecules in solution have an alternative in DNA molecules that are fixed to a surface. Liu et al. [33] take in that biochemical method and develop a molecular computation model to simulate circuits efficiently (the number of parallel operations is proportional to the size of the circuit) [14].

Another possible application of molecular computers is cryptography. In [11] it is outlined how to break the data encription standard (DES) with Adleman's model extended with some more operations. DES codes information with a 56 bit key. For an input pair of cipher text and plain text a classical algorithm had to sequentially test 2^{56} possible keys, what takes some ten thousand years assuming a performance of hundred thousand operations per second. Boneh, Dunworth and Lipton show how to compute the key in about four month of lab work. The *sticker model* by Roweis, Winfree et al. is a computation model that combines DNA manipulations and a random access memory [46]. With one gram of DNA and in spite of many error possibilities DES can be broken by the sticker model in reasonable time [3].

6 Error Resistance

Most of the models mentioned above base on idealized assumptions, in particular deterministic working methods of molecular operations and tests. Indeed the analysis of the biochemical process as well as the repetition of Adleman's experiment that could not deliver definite results [26] let the realization of those assumptions seem to be difficult. The upper bounds of molecular computation models with perfect operations and tests show limits that surely can not be broken by realistic implementation. Nevertheless a profound error analysis and strategies for error resistance help to improve classification of these models and to give statements according their realizability.

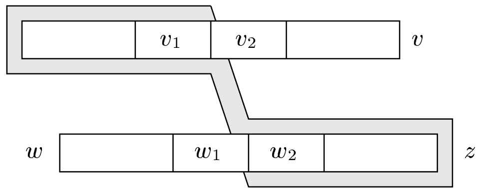
The implementation of the extraction operation with PCR (as for example in [1]) throw doubts on the finite result of a computation. If we assume PCR to extract the proper strands with a probability of 95%, and we undergo only one hundred computation steps, then the probability that exactly those strands are left which meet the extraction criterion is nothing but 0.006% [4]. Instead of that Amos, Gibbons, and Hodgeson implement the extraction operation with the means of restriction enzymes that destroy all strands containing a certain pattern. They reach a distinctly lower error probability than with the use of PCR. The reliability of the extraction operation can also be increased with a proceeding by Karp, Kenyon, and Waarts [27]: they apply PCR to a series of test tubes instead of only one. Boneh and Lipton [13] make their 3-SAT algorithm error resistant by duplicating the number of strands periodically.

Among the first to be engaged in the error analysis of Adleman's and Lipton's operations are Bach, Condon, Glaser, and Tanguay [5]. They increase the probability so when duplicating test tubes indeed all different strands are in both of the tubes, by using a greater amount of equal molecules.

In [7] Baum is concerned with the coding of strings in DNA strands. Using strands of sufficent length it can happen that randomly complementary strands join and so cause errors. A clever string coding takes remedial measures.

7 Splicing Systems

Finally we should not miss to consider a related disposition from the theory of formal languages. An article of Head from 1987 [24] leads this concurrency off. He constructs a language theoretical model after a biological example: a language is given by a test tube filled with DNA and a number of valid operations (realizable with certain enzymes). It consists of mulecular coded strings – the molecules are from the initial set or come off it by enzyme reactions. The formalization of this idea is called *splicing system*, consisting of an initial set of strings on a final alphabet and a rule set of four tuples $(v_1, v_2; w_1, w_2)$ of strings over that alphabet. The rules are applied on two strings v and w at a time. If v contains the pattern $v_1 v_2$ and w the pattern $w_1 w_2$, a new string z is produced, beginning like v including v_1 and ending like w from w_2 on.



The *language generated by a splicing system* is the set of all strings that can be produced by a repeated application of the rules on the initial set. In the theory of formal languages two classes of languages have a great importance: the *class of regular languages* that is easy to decide and the *class of recursively enumerable languages (r. e.)* that not only contains the decidable but also the semi-decidable languages.

Splicing systems have profoundly been investigated. The generative power of splicing systems is examined among others in [24, 38, 40, 41, 21]. Culik and Harju show that a splicing system with regular initial set and finite set of rules generates again only regular sets [17]. But if we allow a regular set of rules, then the set of generated languages is already r. e. [39]. Denninghoff an Gatterdam introduce the *multiplicity of strings in sets* [19]: the appearence frequency of a string in a set is considered. By this means we implicitely get the possibility to add, subtract and multiply integers. If the multiplicity of sets is treated, splicing systems with a finite set of rules can already generate r. e. Also the existence of a "universal computer" in form of a splicing system is proved: there are universal splicing systems that can simulate the working method of any arbitrary splicing system, and they can have regular rule sets or finite rule sets with multiplicity or with an adding mechanism [19, 39, 15]. That means in particular that splicing systems have the same power as Turing machines.

A variation to linear structures as presented above are splicing systems on circular strings without beginning or end. In [25, 47, 42, 52] such cyclic splicing systems are considered, and it turns out that in this case the result of Culik and Harju does not hold [47]. Some extensions enable the construction of universal splicing systems with finite sets of rules that do not need mulitiplicity [52]. Freund looks at splicing systems on graphs in [20].

A combination of splicing systems with the meaning of Head (splicing operations are applied in test tubes) and DNA-computers in the sense of Adleman (new test tubes are filled by extraction from old ones) are examined in [16, 18]. Those *distributed splicing systems* with finite initial and rule set already characterize r. e. On its base a universal computer can be constructed, too.

8 Final Remark

During the last two and a half years many different DNA models have been developed and analysed under several points of view. It turned out to be quite fruitful that many researchers with different interests have picked up that theme. Thereby complexity, stability and realizabilty played an important part.

Even though this chapter of research is still in the beginning. The results up to now show, that DNA-computers will not turn our insights of efficient computation from upside down. Nevertheless they are a powerful instrument for the implementation of parallel processes.

The future of molecular machines could be in a combination with classical computers. Such systems should profit by the specific suitability of the components: well parallizable tasks should be computed with DNA, whereas inherently sequential jobs should be done by silicon-based chips. It is still too early to expect high efficient realizations, but also in biology a unremittingly search for possibilities for a quick and low cost manipulation of molecules is in process.

References

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

[2] L. M. Adleman. On constructing a molecular computer. In E. B. Baum and R. J. Lipton, editors, *DNA based computers*. American Mathematical Society, 1996. Number 27 in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science.

[3] L. M. Adleman, P. W. K. Rothemund, S. Roweis, and E. Winfree. On applying molecular computation to the data encryption standard. In *Proceedings of the Second Annual Meeting on DNA based Computers*, 1996.

[4] M. Amos, A. Gibbons, and D. Hodgson. Error-resistant implementation of DNA computations. In *Proc. of 2nd Annual Meeting on DNA Based Computers*, June 1996.

[5] E. Bach, A. E. Condon, E. Glaser, and C. Tanguay. DNA models and algorithms for NP-complete problems. In *Proc. of 11th Conference on Computational Complexity*, pages 290–299, 1996.

[6] E. B. Baum. Building an associative memory vastly larger than the brain. *Science*, 268:583–585, April 1995.

[7] E. B. Baum. DNA sequences useful for computation. Technical report, NEC Research Institute, 1996.

[8] E. B. Baum and D. Boneh. Running dynamic programming algorithms on a DNA computer. In *Proceedings of the Second Annual Meeting on DNA based Computers*, 1996.

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

[10] D. Beaver. Molecular computing. Technical report, Penn State University, January 1995.

[11] D. Boneh, C. Dunworth, and R. J. Lipton. Breaking DES using a molecular computer. Technical report, Princeton University, 1996.

[12] D. Boneh, C. Dunworth, R. J. Lipton, and J. Sgall. On the computational power of DNA. Technical report, Princeton University, 1995.

[13] D. Boneh and R. J. Lipton. Making DNA computers error resistant. Technical report, Princeton University, 1996.

[14] W. Cai, A. E. Condon, R. M. Corn, E. Glaser, Z. Fei, T. Frutos, Z. Guo, M. G. Lagally, Q. Liu, L. M. Smith, and A. Thiel. The power of surfaced-based DNA computation. Technical report, University of Wisconsin, July 1996.

[15] E. Csuhaj-Varjú, R. Freund, L. Kari, and G. Păun. DNA computation based on splicing: universality results. In *Proceedings of the First Annual Pacific Symposium on Biocomputing*, 1996.

[16] E. Csuhaj-Varjú, L. Kari, and G. Păun. Test tube distributed systems based on splicing. *Computers and AI*, 15(2–3):211–232, 1996.

[17] K. Culik II and T. Harju. Splicing semigroups of dominoes and DNA. *Discrete Applied Mathematics*, 31:261–277, 1991.

[18] J. Dassow and V. Mitrana. Splicing grammar systems. *Computers and AI*, 15(2–3), 1996.

[19] K. L. Denninghoff and R. W. Gatterdam. On the undecidability of splicing systems. *International Journal of Computer Mathematics*, 27:133–145, 1989.

[20] R. Freund. Splicing systems on graphs. In *Proceedings of Intelligence in Neural and Biological Systems*, pages 189–194, May 1995.

[21] R. W. Gatterdam. Splicing systems and regularity. *International Journal of Computer Mathematics*, 31:63–67, 1989.

[22] F. Guarnieri, M. Fliss, and C. Bancroft. Making DNA add. *Science*, 273:220–223, July 1996.

[23] J. Hartmanis. On the weight of computations. *Bulletin of th European Association for Theoretical Computer Science*, 55:136–138, February 1995.

[24] T. Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinat behaviours. *Bulletin of Mathematical Biology*, 49:737–759, 1987.

[25] T. Head. Splicing schemes and DNA. *Nanobiology*, 1:335–342, 1992.

[26] P. D. Kaplan, G. Cecchi, and A. Libchaber. Molecular computation: Adleman's experiment repeated. Technical report, NEC Research Institute, 1995.

[27] R. Karp, C. Kenyon, and O. Waarts. Error resilent DNA computation. Technical report, École Normale Supérieure de Lyon, September 1995.

[28] S. A. Kurtz, S. R. Mahaney, J. S. Royer, and J. Simon. Active transport in biological computing (preliminary version). In *Proceedings of the Second Annual Meeting on DNA based Computers*, 1996.

[29] T. H. Leete, M. D. Schwartz, R. M. Williams, D. H. Wood, J. S. Salem, and H. Rubin. Massively parallel DNA computation: Expansion of symbolic determinants. In *Proceedings of the Second Annual Meeting on DNA based Computers*, 1996.

[30] R. J. Lipton. Speeding up computations via molecular biology. Technical report, Princeton University, 1994.

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

[32] R. J. Lipton. Using DNA to solve NP-complete problems. Technical report, Princeton University, 1995.

[33] Q. Liu, Z. Guo, A. E. Condon, R. M. Corn, M. G. Lagally, and L. M. Smith. A surface-based approach to DNA computation. In *Proceedings of the Second Annual Meeting on DNA based Computers*, 1996.

[34] D. A. Mac Dónaill. On the scalability of molecular computational solutions to NP problems. *The Journal of Universal Computer Science*, 2(2):87–95, February 1996.

[35] M. Ogihara. Breadth first search 3SAT algorithms for DNA computers. Technical report, University of Rochester, July 1996.

[36] M. Ogihara and A. Ray. Simulating boolean circuits on a DNA computer. Technical report, University of Rochester, August 1996.

[37] J. S. Oliver. Computation with DNA-matrix multiplication. In *Proceedings of the Second Annual Meeting on DNA based Computers*, 1996.

[38] G. Păun. On the power of the splicing operation. *International Journal of Computer Mathematics*, 59:27–35, 1995.

[39] G. Păun. On the splicing operation. *Discrete Applied Mathematics*, 70(1):57–79, September 1996.

[40] G. Păun. Regular extended H systems are computationally universal. *Journal of Automata, Languages and Combinatorics*, 1(1):27–36, 1996.

[41] G. Păun, G. Rozenberg, and A. Salomaa. Computing by splicing. *Theoretical Computer Science*, 168(2):321–336, 1996.

[42] D. Pixton. Linear and circular splicing systems. In *Proceedings of Intelligence in Neural and Biological Systems*, pages 181–188, May 1995.

[43] D. Rooß and K. W. Wagner. On the power of bio-computers. Technical report, Universität Würzburg, February 1995.

[44] D. Rooß and K. W. Wagner. On the power of DNA computing. *Information and Computation*, 131(2):95–109, December 1996.

[45] P. Rothemund. A DNA and restriction enzyme implementation of Turing machines. In E. B. Baum and R. J. Lipton, editors, *DNA based computers*. American Mathematical Society, 1996. Number 27 in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science.

[46] S. Roweis, E. Winfree, R. Bourgoyne, N. V. Chelyapov, M. F. Goodman, P. W. K. Rothemund, and L. M. Adleman. A sticker based architecture for DNA computation. In *Proceedings of the Second Annual Meeting on DNA based Computers*, 1996.

[47] R. Siromoney, K. G. Subramanian, and V. R. Dare. Circular DNA and splicing systems. In *Proceedings of the 2nd International Conference on Parallel Image Analysis*, pages 260–273, 1992.

[48] W. P. C. Stemmer. The evolution of molecular computation. *Science*, 270:1510, December 1995.

[49] K. W. Wagner. *Einführung in die Theoretische Informatik, Grundlagen und Modelle*. Springer-Verlag, 1994.

[50] R. M. Williams and D. H. Wood. Exascale computer algebra problems interconnect with molecular reactions and complexity theory. In *Proceedings of the Second Annual Meeting on DNA based Computers*, 1996.

[51] E. Winfree. On the computational power of DNA annealing and ligation. In E. B. Baum and R. J. Lipton, editors, *DNA based computers*. American Mathematical Society, 1996. Number 27 in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science.

[52] T. Yokomori, S. Kobayashi, and C. Feretti. On the power of circular splicing systems and DNA computability. Technical Report CSIM 95-01, University of Electro-Communications, Tokyo, July 1995.