Molecular Computation and Splicing Systems

J.H.M. Dassen[1]

August 30, 1996

[1] Author's current address: Department of Computer Science, Leiden University, P.O. Box 9512, 2300 RA Leiden, The Netherlands; jdassen@wi.LeidenUniv.nl.

Abstract

This thesis provides an overview of the related subjects of Molecular Computation and Splicing Systems.

Molecular Computation is computation using (biological) macromolecules like DNA as information carriers. These macromolecules are manipulated using biological operators, such as enzymes, and operations commonly used in bio-technology and genetic manipulation, such as filtering operations. It has received much attention following Adleman's seminal article [Adl94].

Splicing Systems are models in Formal Language Theory that use the splicing operator instead of concatenation. The splicing operator is an operator on two strings that is an abstraction of the effect of restriction enzymes on strands of double-stranded DNA combined with ligation. It was introduced in [Hea87].

Several models are discussed, including models that are capable of universal computation. Biochemical background is provided in an appendix. Extensive references are provided.

Bibliographic data

```
@mastersthesis{Dassen96,
  author = {J.H.M. Dassen},
  title = {Molecular Computation and Splicing Systems},
  school = {Department of Computer Science, Leiden University},
  address = {P.O. Box 9512, 2300 RA Leiden, The Netherlands},
  month = {August},
  year = {1996},
  url = {\url{ftp://ftp.wi.LeidenUniv.nl/pub/CS/MScTheses/dassen96.ps.gz}},
  homeurl = {\url{http://www.wi.LeidenUniv.nl/~jdassen/}},
}
```

To my parents, who have always helped and stimulated me in finding my way.

Foreword

The research that resulted in this thesis was planned to produce a critical overview of the theory of Splicing Systems. One of the first papers I studied mentioned Adleman's work on solving in the laboratory a problem using DNA as an information carrier. This fascinating work caused a change of plan, broadening the subject to include Molecular Computation.

The huge amount of papers dealing with Molecular Computation that were published (or made accessible to the research community on-line) in the less than two years since the publication of Adleman's paper forced me to make a selection, which I hope has not distorted the overview I'd like to provide too much and which represents the balance between theory and practice I feel comfortable with.

I thank all those who helped me during the research for and the writing of this thesis: Grzegorz Rozenberg, my thesis advisor, who allowed me to change from the original plan; Hendrik Jan Hoogeboom, my secondary thesis advisor, upon whom circumstances forced much more work than is customary for that role; Nikè van Vugt, who provided me with early, fast feedback; Jurriaan Hage, Egbert Boers, Reinier Balt, , critical proofreaders who offered valuable suggestions; Remko Buitenhuis, for checking the appendix; Len Adleman, for getting me on the DNA computing mailing list and for providing me with archives of that list; Samuel Braunstein, for sending me a copy of his tutorial on Quantum Computation; and finally to everyone who made the Internet the invaluable research tool it is today and who made the freeware tools used in editing, typesetting and managing this document.

Ray Dassen — Leiden, August 1996

Contents

Chapter 1

Introduction

1.1 A definition of molecular computation

Molecular computation is computation using biological macro-molecules like DNA as information carriers, that are manipulated using biological operators, such as enzymes, and operations commonly used in bio-technology and genetic manipulation, such as filtering operations and the polymerase chain reaction.

Like many recent developments in Computer Science, such as genetic algorithms and neural networks, Molecular Computation is inspired by nature. While these recent developments mostly use nature as a resource to inspire new techniques for problem solving, as a basis for software, Molecular Computation focuses on nature as a basis for hardware.

1.2 Adleman's approach

The current boom in interest in molecular computation started with Adleman's seminal article [Adl94], in which an instance of the directed Hamiltonian path problem — a well-known NP-complete problem — is solved using DNA and standard biological techniques.

Adleman's approach is based on a nondeterministic algorithm for the directed Hamiltonian path problem. The algorithm amounts to the following: generate a random path through the graph and apply a (polynomial-time) verification algorithm (see [CLR90]) to it. The path is encoded as a DNA sequence, and the verification algorithm is implemented by sequentially checking the properties of the sequence encoding a candidate solution path and discarding that path if it does not fulfill the requirement being checked.

Adleman shows how to solve instances of one specific combinatorial problem, Hamiltonian Path, in a way that easily suggests a generalization to other combinatorial problems. This generalization was first formalized by Lipton ([Lip94, Lip95b, Lip95a, Lip96]), who provided clear definitions of data representation, initialization and operations.

We will study Adleman's approach and Lipton's generalization in detail in Chapter 3.

1.3 Why yet another model?

There are a number of reasons why Molecular Computation is interesting from both a theoretical and a practical viewpoint.

As a study of complex systems, Computer Science has many large and often fuzzy borders with Mathematics, Physics, Biology, Chemistry, Linguistics and other sciences. These sciences frequently use Computer Science to implement and study their models on computing devices. Some of these models have led to 'architectures', models of computation, either in hardware or as theoretical constructs, like neural networks, classifier systems, Post production systems, string rewriting systems, Turing machines, cellular automata, spreadsheets etcetera.

From the viewpoint of computability, one sufficiently strong model, say Turing machines, is enough. For every algorithmic device of all other known and unknown architectures, there exists a Turing machine that performs the same computation as that device. This unprovable, but generally accepted statement is known as the Church-Turing hypothesis.

Indeed, all computations of every device in all of the models can be simulated by a single, so called *universal* Turing machine. The importance of constructing a universal machine (either physically or as a mathematical construct) is that the universal machine is programmable: it in itself is enough to perform all computations, meaning that from the viewpoint of computational strength, no other hardware needs to be built. In all of the sufficiently strong — i.e. Turing machine equivalent — models, there exist universal devices.

There are quite a few reasons why it is important to study more than one single model and to study weaker models too, though.

Alternative viewpoints. One model can be a lot more natural for expressing a problem than another. This has been noted many times. In linguistics it is known as the Whorf/Sapir hypothesis ([Who40, Sap58]) expressed colloquially as 'Language shapes thought'. [Bea94] formulates it thus: 'If all you have is a hammer, everything looks like a nail'. For instance, we normally view a computer as 'a universal Turing machine with limited tape'. This model provides us with a structure suitable for programming purposes, which the formally equivalent model under the limited tape assumption of finite state machines lacks.

Formality and ease of use. Different models often represent different choices in the trade-off between formality and ease of use. Although Pascal is much easier for practical programming purposes than Turing machines, its semantics are less formally defined, and attempts to develop or use formal semantics for it are daunting tasks.

Ease of implementation. Also, for a specific kind of hardware, one model is easier to implement than another. Writing a compiler for an Algol-like language on current electronic computers is more straight-forward than writing a compiler or interpreter for a functional language. On the other hand, due to their lack of side-effects, purely functional languages can easier exploit parallel architectures than can imperative ones.

Applicability of theories from other sciences. Using a different model one can sometimes use theories from other sciences to perform analysis on practical aspects. For instance, statistical mechanics have successfully been used in analysing the behaviour of neural networks.

Differences in what problems are tractable. Furthermore, although most of the models can simulate each other using a simulation of polynomial time-complexity, there are indications (e.g. [Sho94]) that some implementable models (especially quantum computation) may be more than polynomially faster than others, thus possibly enlarging the range of problems that can be considered tractable.

By providing different viewpoints, having multiple models improves our insight in computability and the essence of algorithms and complexity.

Hierarchies: strength versus analysability. The different models can be weakened in different ways, thus inducing hierarchies (actually partially ordered families of classes of algorithms) that describe different positions in the trade-off between computational power (or expressibility) and suitability for analysis (or decidability or effectiveness of decision problems): more can be proved for less powerful models and the decision algorithms for the weaker model are often less complex than for the stronger model.

The classical example of such hierarchies is the Chomsky hierarchy (see [HU79, Ch. 9]) consisting of the regular languages, the context-free languages, the context-sensitive languages and the recursively enumerable languages. All of these languages are in practical use: regular languages describe the patterns that the Unix `grep(1)` command can recognize; context-free languages are used in describing the syntax of programming languages; context-sensitive languages have applications in natural language processing and linguistics; and recursively enumerable languages describe the limits of what programs can compute.

1.4 Advantages and promises of Molecular Computation

Molecular Computation (also known as 'DNA computation') is a fairly recent model, with many promises, both practical and theoretical.

Molecular Computation is based on nature's way of generating complexity: the interaction between DNA, the carrier of genetic information, and enzymes (operators on DNA). It has potential for easily allowing massive parallelism on an unseen-before scale, providing new insights for biology, pharmacy, and medicine. It has already resulted in a new branch of formal language theory, using splicing operators that are modeled after enzyme operations, instead of concatenation.

Energy-efficient. Nature is highly efficient in its energy use: only a small percentage of energy produced is in a useless form. Cells routinely achieve the same reaction speeds at lower temperatures — thus wasting less energy on heat — as our chemical technology, because the enzymes they have evolved are such good and selective catalysts. In electronic computing too, much energy is wasted as heat[1] and subsequently on cooling. Using nature's toolbox can make molecular computers much more energy-efficient than previous types of computers.

Molecular Computation will push biology and biochemistry for more flexible operations and more reliable techniques that may prove useful in those disciplines and in medicine. Its focus on computation, on information storage, retrieval and manipulation may also inspire new insights into the early phases of the evolution of life on earth.

Massive parallelism: a nondeterministic computer. A sequential computer is an approximation of a deterministic Turing machine, where the unbounded tape is approximated by a large, but finite memory. In the same way, a parallel computer is an approximation of a nondeterministic Turing machine where the unbounded number of parallel computational searches is approximated by a large, but finite number of parallel searches. Adleman's approach has attracted a lot of attention, because the number of parallel searches it permits is larger than that of previous parallel computers by several orders of magnitude. From the viewpoint of computational power, it is nothing new ('yet another non-deterministic Turing machine'), but from a practical perspective it is very interesting, since it may redefine the limits of feasible computation.

Density of information storage. The potential density of information storage is very high: 66–67 atoms per base pair (including the sugar-phosphate backbone and supporting ions); each base pair can encode for four symbols (two bits).

1.5 Limitations and drawbacks of Adleman's approach

There are a number of drawbacks and limitations to Adleman's approach, some of which have been resolved.

Adleman's approach is designed to solve combinatorial problems only. No attempt is made to achieve universal computation.

[1] Some modern CPUs can be used to bake eggs on when their cooler is removed.

Furthermore, the operations involved are very slow and highly error prone compared to those in digital electronic computers, and the model's scalability to large problem instances is doubtful.

Also, it requires an external operator to perform the various filtering steps; it cannot be 'switched on and run by itself'.

We will review these and other problems in more depth in Chapters 3 and 6.

Research since has focused on removing or alleviating these limitations. There are now several universal models; some approaches do not require an operator; and less error prone operations, as well as probabilistic approaches to increase reliability are being studied.

1.6 Other approaches to molecular computation

More general models have been proposed, such as Lipton's model for solving NP-complete problems. We will focus on this model in Section 3.2. Other researchers have focused on achieving universal molecular computation, by moving away from the use of DNA as a static 'write-once' representation of a potential solution to a dynamic use of DNA as a more general information carrier, manipulated using enzymes.

The striking similarity of DNA to Turing machine tape has been exploited to simulate Turing machines, including universal ones. Of these, we will discuss Beaver's (Section 4.2) and Rothemund's (Section 4.3).

A different approach was taken by Winfree, who studied non-linear DNA sequences generated by self-assembly in [Win95b, WYS96]. These allow trees to be represented directly, and seem to be capable of universal computation by themselves (without manipulation by enzymes): one-dimensional cellular automata can be simulated by two-dimensional DNA 'clusters'. This approach is briefly discussed in Section 4.4.

Chapter 2

Classifying models of Molecular Computation

Molecular Computation is a quite new discipline, and still has a way to go before becoming a mature technology. It has been compared in [Smi95] to the vacuum tubes phase in the development of electronic computers. Befitting this stage, the number of radically different models is large, and it is difficult to tell yet which models are viable and which will be culled by implementation problems or economic feasibility.

This state of affairs makes it hard to give a precise definition that includes all of the models. It is more illuminating to focus on and clarify the differences between the various models.

We will therefore list the important differences. Please note that most of the distinctions made here are formulated as dichotomies, although practice will show many shades of gray between black and white. Also, the distinctions are not to be taken as orthogonal: many of them are interrelated, often strongly.

2.1 Special purpose or universal

Some models are designed to handle only a limited class of problems, mostly those that are intractable using electronic computers, such as problems involving large combinatorial searches. Examples of these models are Adleman's and Lipton's approach of combinatorial searches encountered when attacking NP-complete problems ([Adl94, Lip94, Lip95a]) and problems in cryptography ([Bea94, BDL95, ARRW96, BDL96]).

Other models focus on achieving universal Molecular Computation. A universal model is important, because in a universal model there exist universal members: members that can be *programmed* to perform any computable task. Most of these models are proof-of-concepts rather than practical proposals, although most researchers express optimism about achieving practical universal Molecular Computation. These models include Beaver's and Rothemund's simulations of Turing machines — either generic ones or a universal one specifically — which we will study in Chapter 4 and Kurtz' RNA editing and rewriting based one ([KMRS96]).

2.2 In vitro or in vivo

Most current models are *in vitro*. *In vitro* models are conceptually less complex than *in vivo* ones[1], but *in vivo* models may have several advantages. Cells have evolved high reaction speeds and efficient ways of separating reactions by selective transportation of reaction products and disposal of byproducts. Even if *in vivo* Molecular Computation may be infeasible, living systems are an important source for tricks and techniques. For instance, the use of multiple tubes can be seen as an imitation of the cell's compartments.

2.3 The information carrier

The various models of Molecular Computation employ different information carriers.

Most schemes use DNA, but DNA has many forms in which it is suitable. Single-stranded and double-stranded DNA are common, but more exotic forms are used too, such as circular DNA (modelled theoretically in [YKF95]), partially double-stranded DNA ([RWB$^+$96]), non-linear DNA ([WYS96]). However, double-stranded DNA appears to be the preferred form under most reaction circumstances, and the assembly of 'unusual' DNA structures is still more a topic of research than a technology ([SWL$^+$96]).

[1] *In vivo* models are those models in which the computation is carried out in the biochemistry of a living organism, such as a genetically modified *Escherichia Coli* bacterium.

Some researchers have speculated on the use of artificial DNA-like polymers such as PNA, either as a more stable information carrier[2] or for intermediate use in operations ([RWB$^+$96]).

RNA is also suitable as an information carrier. In nature, RNA is more often manipulated and rewritten than DNA. DNA's function is mostly that of a rather passive, nearly read-only, long-lived instruction tape, while RNA is more of a short-lived scratch memory used in protein synthesis. RNA is the main inspiration for the abstract CNA model in [KMRS96].

2.4 The operations

Current models employ a wide range of operations on information carriers from areas like physical chemistry and biotechnology, such as filtering, magnetic bead extraction and gel electrophoresis, and biochemistry biology, including enzymatic reactions such as PCR.

The choice of operations in a model depends on several factors. One is the function of the information carrier in the model, which we will discuss shortly. Another is — of course — the information carrier itself.

The scalability of models is influenced by the dimensionality involved in the operations: some operations are volume-based (e.g. the addition of enzymes), while others — especially magnetic bead extraction and gel electrophoresis — use less than three dimensions, and may therefore scale less well.

A factor that is currently receiving much attention, is the reliability of operations. There are many examples of unreliable behaviour. Filtering operations may not filter out all undesired molecules, and not pass all desired ones. Ligation also happens between strands that are not fully Watson-Crick-complementary. PCR — often used to extract (partial) solutions — may boost undesired molecules too, and can produce distorted copies; also, some reaction circumstances favourable for PCR can result in reactions between the molecules to be copied ([KCL96]).

[2] Though no enzymes for manipulating PNA exist. ([Smi95])

2.5 Are instructions data?

A universal computer is programmable: it can execute every algorithms given an encoded description of it — a program — as part of its input. In the development of electronic computers the realization that instructions are not fundamentally different from data, and thus can be treated as data themselves, was crucial[3]: instead of rewiring the hardware to perform a different function, one could simply load a new program tape.

In Molecular Computation only a few models, such as Rothemund's Turing machine simulation which we will study in Chapter 4, treat instructions as data. In most models, instructions are either interwoven in the hardware or part of the operator (e.g. an external robot that mixes tubes).

2.6 Manipulation of the information carrier itself

In Adleman's and similar models, the information carrier is not rewritten after an initialization phase. It merely encodes a candidate solution in an exhaustive search.

In other models, the information carriers may be rewritten like the tape in a Turing machine or RAM in an electronic computer. This allows the model to overcome the limitations of the exhaustive search approach. An obvious example is in Turing machine simulations, where the DNA that simulates the tape is rewritten. But the complexes formed in simulating the development of cellular automata ([Win95b, WYS96]) fall into this category too.

2.7 One-pot or multiple phases or tubes

An important problem in Molecular Computation is that of control of operations. While it is desirable in terms of speed to achieve a high degree of parallelism between the operations in performing molecular programs, actually performing operations in parallel can be expected to result in many undesirable interactions between the operators and operands of the various operations.

[3] It is one of the achievements of John von Neumann, who created the basic architecture of electronic computers that is still used today.

One way to prevent this is simply to perform operations sequentially, and clean up between them[4]. A big disadvantage is that, because one has to be on the safe side of the duration of the reactions involved in the operations, the worst case times of the reactions add up. In the parallel case, the worst case is that of a pipeline of operations, which may be significantly shorter than the worst cases of the individual operations added up.

A different approach is to perform the different steps in different tubes or vats, separated by filters, membranes or purifying operations to control the transport of molecules between them. [RWB$^+$96] presents a discussion model of a 'parallel robotic workstation for molecular computation' that uses membranes.

An approach which is designed for parallelism and for which no external operator is necessary after an initialization phase is termed 'one-pot'. There are currently no 'one-pot' schemes based on well-understood chemistry. The closest yet are Winfree's cellular automata [Win95b, WYS96].

2.8 Error-resilience

Models differ widely in their attention for errors. Some simply assume perfect operations, while others are designed to be resilient to errors. The approaches to error-resilience include refinery based on biased random walks ([RWB$^+$96]) and repeating operations based on probabilistic analysis ([KKW95]), transforming algorithms to use error prone operations less frequently ([KKW95, AGH96]), selective amplification of 'good' strands ([BDLS96]), double encodings ([BDLS96]) and stricter designed, less dense encodings ([Mir96, DMG$^+$96, Bau96b]).

2.9 Communication

Currently, models have no communication between information carriers, and communication to an external operator is often limited to an end result readout (the operator manipulates 'blindly').

Researchers are however very interested in developing models that exploit various forms of communication ([Ame96b]). For instance, branched parallel

[4] Fast cleanup can be achieved by fixing the information carrying molecules on solid support, and cleaning up by washing. Examples are [Rot96, LGC$^+$96].

search could potentially be done much deeper when ‘dead end’ strands could be reused for exploration of promising paths.

2.10 Native or not

Current universal models of Molecular Computation are all based on simulating or implementing previously known abstract universal models. As yet, there is no model that is ‘native’ to Molecular Computation, one that fits seamlessly into the potential offered by Molecular Computation, while avoiding its disadvantages.

For current electronic computers, the register machine is a ‘native’ model. ‘native’ models are conceptually easier to implement, because they fit better than ‘alien’ ones, that have to be translated before becoming useful.

Chapter 3

Special purpose models

3.1 Adleman's approach

In [Adl94] Adleman used techniques from molecular biology to solve a 7-node instance of the Directed Hamiltonian Path problem (DHPP).

3.1.1 The Directed Hamiltonian Path problem

A Directed Hamiltonian Path in a graph is a directed path through that graph that starts in a designated node, ends in another designated node, and passes all other nodes exactly once. The DHPP is known to NP-complete[1]. It is therefore highly likely that no algorithm for this problem exists that has less than exponential time-complexity. Furthermore, since many NP-complete problems are of practical interest and algorithms for solving one NP-complete problem can be transformed – adding only polynomial factors – into algorithms for another, efficient methods for solving even one of them are very desirable.

In one formulation, the problems in NP are those problems for which a polynomial time verification algorithm exists. In most of the interesting ones, the number of candidate solutions is exponential (or worse) in the problem

[1] For an overview of the theory of NP-completeness, the reader is referred to [GJ79]. [CLR90] provides an overview of the field of algorithm analysis, including time-complexity and NP-completeness.

size. A brute force solution is to generate all possible candidate solutions, and apply the verification algorithm to each of them. This is the approach taken by Adleman: the whole brute force search is performed in parallel.

3.1.2 The algorithm

The input to an algorithm for DHPP is a graph. There are two possible types of output: a simple 'Yes' or 'No' answer to the question 'does this graph have a Hamiltonian Path'[2] or such a path, if one exists.

Adleman's method is an implementation of the following algorithm:

1. Generate 'all' paths through the graph.
2. Discard paths that do not start in the designated start node or that do not end in the designated end node.
3. Discard all paths that do not consist of n nodes (n being the number of nodes in the graph).
4. Retain only paths in which each node appears at least once.
5. If any paths remain, a solution is found. If not, there is no solution.

3.1.3 Implementation

The key to the implementation is the representation of the edges in the graph. This representation is based on that of the nodes. Each node i is encoded as a 20 base pairs long sequence of DNA[3], designated O_i. The encoding $O_{i\rightarrow j}$ of an edge $i \rightarrow j$ consists of the 10-base tail of O_i followed by the 10-base head of O_j. We use a bar to indicate the Watson-Crick-complement of a sequence, e.g. $\overline{O_3}$. When ligating with $\overline{O_i}$ or $\overline{O_j}$, this encoding ensures 10-base sticky ends: the strand encoding the edge $i \rightarrow j$ and the strands encoding the nodes i and j are connected in a brick-like manner.

[2] This is the so called *decision problem* version; the theory of NP-completeness is formulated for decision problems only.

[3] [Adl94] states 'a random 20-mmer sequence of DNA'. This makes the algorithm more probabilistic than necessary. With some effort, sequences can be chosen, without resorting to a probabilistic method, that have the desired qualities and the additional benefit of lowering the chances of erroneous extraction and self-stickiness. ([Bau96b])

In step 1, 'all' paths through the graph are generated. From step 3, one can see that it is sufficient to generate all paths of length n; the mechanism used here will produce other paths, that are filtered out in step 3. Generating all these candidates requires super-exponential time on a deterministic computer[4], but can be done efficiently using parallel chemical reactions. For all nodes i except the designated start and end node and all edges $i \rightarrow j$, $O_{i \rightarrow j}$ is ligated with $\overline{O_i}$ and $\overline{O_j}$. This results in chaining $O_{* \rightarrow i}$ to $O_{i \rightarrow j}$ to $O_{* \rightarrow i \rightarrow j \rightarrow *}$.

Step 2 is done by PCR amplification using O_{start} and $\overline{O_{\text{end}}}$ as primers, resulting in amplification of those molecules that encode paths with the desired start and end node.

The separation on length for step 3 is done using gel extraction. In gel extraction, the molecules are run through a gel. The speed with which a molecule runs through the gel depends on its weight. The molecules are thus sorted in bands of molecules with the same weight. The molecules in the $20n$ band (corresponding to paths with length n) are excised from the gel to be used in the next step.

Step 4 is done as follows. The product of step 3 is purified first, and single-stranded DNA is generated from it. Next, repeated extraction — once for each node i — is done using a magnetic beads system: $\overline{O_i}$ conjugated to magnetic beads is added, allowed to anneal to strands containing O_i. These are extracted, and retained for the next iteration.

Step 5, the readout, is done by first PCR amplifying the remaining solution, and detecting the presence of DNA using gel electrophoresis. This answers DHPP as a decision problem. If a solution path is desired, one of the strands detected can be sequenced.

3.1.4 Classification

Adleman's approach is a special purpose *in vitro* model, using DNA (both single-stranded and double-stranded) as an information carrier; selection is done using PCR, gel electrophoresis, and magnetic bead extraction. The information carrier is not rewritten after the initialization phase and is processed in multiple physically separated phases. Several sources of errors are identified, but the implementation was not designed to be very error-resilient.

[4] Taking into account the start and end node, and the fact that each node has to appear precisely once, the number of candidate solutions is still $(n-2)!$.

3.1.5 Results and problems

Implementation problems. Adleman did find a solution for his chosen 7-node instance of the Directed Hamiltonian Path problem, with approximately 7 days of lab work. He identifies several sources of errors that have to be investigated when upscaling his approach:

- The ligation used in step 1 to produce the paths through the graph may also occur between incompatible edge oligonucleotides. The resulting molecules look somewhat like paths, but do not encode for actual paths. It is therefore advisable to check if the outcome of the computation indeed encodes an actually occurring Hamiltonian path.
- The separation in step 4 is not perfect: 'good' molecules may not be extracted, while some 'bad' ones are.
- PCR has since been shown to be a severe source of errors unless care is taken ([KCL96]), because the single stranded templates to be copied can anneal to each other.

Advantages. On the positive side, Adleman notes that the encoding used has several advantages. It is unlikely that long common subsequences between codings for different vertices would exist, which could result in more 'unintended' binding during step 1. Also, unusual features like hairpin loops are unlikely to appear. Furthermore, the length of the oligonucleotides used is such that they are stable at room temperature.

Limitations of the abstract model. While the problems just discussed are mostly associated with the implementation, there are limitations to the abstract model too. The main limitation is that it cannot break the exponential barrier: exponentially complex algorithms remain infeasible even for fairly small problem instances. [Har95b] shows that solving a 200 node instance of DHPP would require an amount of DNA weighing more than the Earth. [MD96] uses a similar argument — using an estimation of the amount of matter in the universe — to show that the increase in the size of generally solvable problems for which only exponential algorithms are known achievable via Adleman-like models is limited to approximately 10^2.[5]

[5] Even stronger: 'Over the chemically measurable picomolar to molar concentration range the greatest practical increase in problem size is limited to $\approx 10^1$'. However, this ar-

Also, the output of the initialization step falls in a limited class of languages. When the self-assembly is linear — as it is in Adleman's approach — this class is that of regular languages ([WYS96]).

3.2 Lipton's model

Adleman's approach to the Directed Hamiltonian Path problem was groundbreaking, primarily because it was shown to actually work in the laboratory (although for a small test-case). An important problem in generalizing the approach of [Adl94] is that algorithm and implementation are strongly interwoven: [Adl94] does not make a clear distinction between the abstract model of data structures with their associated operations and the concrete methods of implementing them.

Applying the principle of separation of concern, Lipton made this distinction in [Lip94, Lip95b, Lip95a]. He provided an abstract model of Adleman's approach by clearly identifying the operations used, thus allowing computer scientists to study the power of this model without going into implementation details and conversely allowing biologists to look for mechanisms to implement the various operations without having to fully understand the way they are used in implementing algorithms.

Lipton illustrates the the usefulness of his abstract model by generalising Adleman's approach to other NP-complete problems, especially the satisfiability problem (SAT).

3.2.1 Operations

Lipton's model introduces the notion of *test tube* — [Lip94]: 'from the point of view of a computer scientist, it is just a finite multiset of strings from $\{A, C, G, T\}$' — and includes the following operations:

Initialize Create a test tube containing large numbers of copies of some short single strands.

Anneal The operation of creating double stranded DNA from complementary single strands.

gument is much weaker, since biological operations such as PCR can be used to selectively amplify chemically immeasurable amounts of DNA up to detectable amounts.

Extract Extracting those DNA sequences that contain a certain consecutive pattern.

Detect Determine if a test tube contains any DNA strands at all.

Amplify Replicate all the DNA strands in a test tube.

Note that all of these operations are present in Adleman's approach: *initialize* and *anneal* in step 1, *extract* in step 4, *amplify* between steps 3 and 4, and *detect* in step 5.

3.2.2 Solving SAT

SAT. *SAT (satisfiability)* is the NP-complete problem of finding an assignment of values to the variables in a boolean formula that that satisfies that formula, i.e. makes it evaluate to *true*. E.g. the solutions for $(a \vee b) \wedge (\bar{a} \vee b)$ are $(a, b) = (true, true)$ and $(a, b) = (false, true)$; for $a \wedge \bar{a}$ there is no solution. Variables like a, and their complements like $\bar{a}$ are known as *satisfiability problem!literals inliterals*.

CNF: a normal form for SAT. A normal form for boolean formulas is *conjunctive normal form (CNF)*, in which each formula is built up as a conjuction (AND) of clauses, each of which is the disjunction (OR) of one or more literals. An example formula in CNF is $(x \vee y \vee z) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z}) \wedge (x \vee \bar{y} \vee \bar{z})$.

Using CNF as a normal form makes it easier to express algorithms for SAT, since formulas in CNF have an easier structure than general boolean formulas. This advantage is especially important to Lipton's algorithm, since the structure of a formula determines the operations to be performed.

Encoding candidate solutions. Candidate solutions for an n-variable instance of SAT can be encoded as n-bit numbers (bit i is one, means x_i is *true*). These n-bit numbers can be encoded as paths through a simple graph. Such a graph has nodes $a_1, x_1, {x_1}', a_2, x_2, {x_2}', \ldots, a_{n+1}$ and edges from a_k to x_k and ${x_k}'$ and from x_k to a_{k+1} and from ${x_k}'$ to a_{k+1}. In such a graph all paths from a_1 to a_{n+1} encode an n-bit binary number: if a path contains the a_k to x_k edge it encodes for a number with bit k as one; if it contains the a_k to ${x_k}'$ edge, bit k is zero. The graphs are encoded into DNA following Adleman's encoding.

The algorithm. The key idea in Lipton's algorithm is that the formula determines which operations to perform and in what order.

ORs are done using multiple tubes, and ANDs are done by repeated extraction in a kind of 'function application': start with all strands, and loop over the clauses, keeping only those strands corresponding to variable assignments that evaluate to *true* for the current clause. After this iteration, the remaining strands are those that correspond to satisfying assignments for the whole conjunction.

Lipton's algorithm for solving instances of SAT for CNF formulas $C_1 \wedge C_2 \wedge \ldots \wedge C_m$ over n variables is as follows:

- *Initialize* and *anneal*: create a tube t_0 with DNA strands that encode all n-bit binary numbers.
- A tube t_k contains exactly the solutions for $C_1 \wedge C_2 \wedge \ldots \wedge C_k$.
- Tube t_{k+1} is created from t_k by a series of *extract*s, one for each variable in C_{k+1}. t_{k+1} is initially empty. For each *extract*, the strands extracted from t_k are added to t_{k+1}, while the remainder of t_k is used in the following *extract* or discarded when there is no following *extract*. The *extract* for a variable v extracts all strands encoding numbers with the bit for v set to 1 if v appears as a literal in C_k. If $\bar{v}$ appears in C_k, the strands with the v bit set to 0 are extracted. Thus t_{k+1} will contain only the strands that were in t_k (i.e. encoded solutions for $C_1 \wedge C_2 \wedge \ldots \wedge C_k$) and that additionally satisfy C_{k+1}.
- Create t_{m+1} and *detect*. A solution to the instance exists if and only if there are any strands left.

3.2.3 Results and problems

Lipton shows how to solve SAT directly instead of via reduction to HPP and suggests that his method is directly applicable to all problems in NP. This does away with the polynomial factors introduced by reductions, which is a significant advantage.

Furthermore, [Lip94] suggests using a molecular computational device as a special purpose co-processor or oracle for performing exponential searches: an electronic/molecular hybrid computer.

The most important contribution is probably the precise identification and terminology of the operations in Adleman's approach. 'Adleman-style computing' could be termed 'Lipton-like models' with equal validity.

The operations still need to be perfect, except for *extract*: if *extract* does not always result in extracting all strands, this can be compensated for by starting with more strands in the initial tube.

Lipton's model was further formalized into 'DNA-Pascal' and subjected to computational complexity analysis by Rooß and Wagner ([RW95]); Amos, Gibbons and Hodgson ([AGH96]) provide a similar model with a less error-prone implementation of the *extract* operation; and Karp, Kenyon and Waarts ([KKW95]) look at error-resilience through probabilistic methods and provide a transformation from Lipton's model that assumes error-free operations to theirs. Boneh, Dunworth, Lipton and Sgall ([BDLS96]) discuss two techniques for error-resistance that can be applied directly to Lipton's algorithm: PCR-amplification after each step to increase the survival probability of 'good' strands and a double encoding to increase the probability of correct *extracts*.

Chapter 4

Universal models

Adleman and Lipton showed that Molecular Computation is quite powerful: it can potentially solve instances of NP-complete problems of hitherto infeasible size.

However, they did not answer the question of universality: does a model in Molecular Computation exist that is capable of simulating all computations, i.e. can we create a molecular computer in the sense in which 'computer' is used nowadays: a programmable device? Of course, physical devices are necessarily finite and thus not truly universal, but some of them, such as electronic computers, are very good approximations that are sufficient for a large number of practical problems.

The answer turned out to be 'Most likely: Yes'. It was found via two ways.

One was from the theory of splicing systems, a branch of Formal Language theory whose models are abstractions of the combination of DNA annealing and ligation combined with restriction enzymes. We will study this theory in Chapter 5.

The other was from several more or less practical proposals that simulate classes of Turing machines using Molecular Computation. Of these, we will discuss two in some detail: Beaver's ([Bea95b, Bea95d, Bea95c, Bea96]) and Rothemund's ([Rot96]). Also we will briefly discuss a different approach by Winfree ([Win95b, WYS96]).

4.1 Turing machines

For extensive treatment of the Turing machine model, the reader is referred to [HU79]. In this section, we will shortly review a few aspects of the Turing machine model, that are relevant to both Beaver's and Rothemund's simulation.

4.1.1 Basic model

Informally, a Turing machine consists of a finite control that stores a transition table and has a current state, a tape of potentially unlimited length divided into cells that are capable of storing one symbol from a finite alphabet, and a read/write head that can move (in one-cell steps) along that tape.

4.1.2 Representation

Instead of dividing the Turing machine model into 'hardware' (finite control and read/write head) and 'magnetic media' (tape) — a division natural when studying electronic computers — in both simulations, the model is divided into constant and variable components. The constant component is the transition table; all other parts — head position, state of the finite control and contents of the tape — are variable.

4.1.3 Configurations

Both models encode the *configurations* (or *instantaneous descriptions*) of Turing machines in DNA and perform computations by simulating the step-relation between configurations.

A configuration of a Turing machine describes the status of a Turing machine computation at a specific point in that computation. It describes the contents of the tape, the position of the read/write-head and the state of the finite control.

Since at any point during the computation, the non-blank part of the tape is finite, a configuration can be represented by a string of finite length, say $\alpha_1 q \alpha_2$ (where α_1, α_2 are strings of tape symbols, and q is the state of the finite control; q also functions to indicate the position of the read/write-head:

the first symbol of α_2 (under the conventions of [HU79], q cannot be confused with a tape symbol)).

Using a suitable encoding of such a configuration, performing one step of the computation amounts to a function application to the configuration that only depends on a small part of that configuration (a *local* change), namely the part comprising the position of the head, the current symbol and the state of the finite control. This key feature is exploited in both models, though through very different implementations.

4.1.4 (Non)determinism

As we have seen clearly in Adleman's approach, in Molecular Computing it pays to use non-deterministic algorithms. Using non-determinism simplifies the conceptualization of algorithms that can exploit parallel architectures. Similarly, it would be useful to exploit this on the level of the architecture itself.

There are methods to implement a non-deterministic Turing machine on a deterministic one: either explicitly simulate the 'forking' of copies whenever the non-deterministic Turing machine has a choice, or start with a large number of deterministic Turing machines with additional random bits on the input tape, that are then used in determining which choice to follow.

4.2 Beaver's model

Beaver simulates deterministic Turing machines by implementing the step relation on instantaneous descriptions via local changes.

The local change can abstractly be modeled as the replacement of a substring $\alpha X \beta$ by $\alpha Y \beta$ in a string $L\alpha X\beta R$, in which α, β, X, Y are sequences of limited length and where neither α nor β occur elsewhere in the string.

4.2.1 A new operator: context-sensitive substitution

Beaver proposes a specific form of site-directed mutagenesis — a small modification in a strand whose location is determined by a specific base sequence — to implement this substitution. Double strands containing $\alpha X \beta$ are converted to single strands. Strands of $\bar{\alpha}\bar{Y}\bar{\beta}$ are added, whose $\bar{\alpha}$ and $\bar{\beta}$ parts anneal to their complements α and β. The result, except the X and $\bar{Y}$ sequences that cannot bind, is then made double-stranded via PCR.

This results in double strands that are properly aligned except for the part between α and β where substitution is to take place. These strands are denatured into single strands again; primers that anneal to $\bar{L}\bar{\alpha}\bar{Y}\bar{\beta}\bar{R}$ – i.e. starts of L and ends of R – are added, and PCR is performed. Now we have two types of strands: double-strands that encode for $L\alpha Y\beta R$ that are to be kept, and single strands $L\alpha X\beta R$, which need to be removed. These undesired single strands are destroyed by cutting with S1 nuclease. Now the substitution is complete.

4.2.2 Implementing the simulation

The simulation is performed as follows. An initial tube containing a population of strands is generated (by sequence generation, Adleman-like 'generation of diversity' to incorporate random bits for implementing non-determinism, if desired, and PCR).

Next, a sequence of separate, substitute and mix back together is performed. After each such step, detection of strands corresponding to configurations in the halt state is performed.

The contents of the tube are separated according to the local sequence that determines (not necessarily totally) the transition. If non-determinism is to be simulated, the tubes can be duplicated using PCR, or can be further separated on the random bits. On each of the tubes, the appropriate substitution (as determined by the transition table), is performed. Next, the step is completed by mixing together all the tubes, and performing detecting for halted tapes.

4.2.3 Problems

There are several problems with Beaver's implementation of the substitution operator, noted in [Bea95c] and [Smi95].

Unintended complexes can form when the $\bar{\alpha}$ sequence of an $\bar{\alpha}\bar{Y}\bar{\beta}$ sequence anneals to a different strand than its $\bar{\beta}$ sibling. This may be prevented by temporary attachment of the strands on which substitution is to take place to solid support. Another solution suggested is to use circular strands, causing unintended complexes to have improper length, and filtering them out using gel electrophoresis. For useful Turing machines, the length of a tape is unpredictable, and this trick cannot be used (unless one is willing to give up on potential unlimited tape length, and a priori choose a maximum length).

PCR is involved twice in every substitution, although it is slow, expensive and error-prone ([Smi95, KCL96]).

The digestion of single stranded DNA is done with S1 nuclease, which can work on double stranded DNA too, and which requires reaction circumstances that can destroy information stored in DNA.

4.3 Rothemund's model

Whereas Beaver's Turing machine simulation is rather sketchy, Rothemund's paper is quite detailed. Rothemund not only addresses technical aspects like representing configurations in DNA, candidate-enzymes for use in implementing the transition steps and the various problems associated with them, sources of error and error resilience, estimates of size, speed and energy use; he also provides backgrounds to Turing machines and Molecular Computation and motivation of the importance of simulating Turing machines.

Like Beaver's construction, the simulation is performed by implementing the single steps from one configuration (or *instantaneous description*) to another. Unlike Beaver, Rothemund uses a number of enzymes to implement the step relation.

Instead of developing a simulation of a universal Turing machine directly, Rothemund uses a small non-universal Turing machine (a solution to the Busy Beaver problem for three states) as a running example and then suggests how to scale the construction up to a universal Turing machine.

4.3.1 Useful enzymes

A number of enzymes known as *restriction endonucleases* or *restriction enzymes* are used by bacteria to destroy double-stranded foreign DNA. The foreign DNA is recognized by specific words — with a typical length of 6 to 8 bases — forming a *recognition site* (or *restriction site*). The bacteria's own DNA containing these words is protected by a chemical modification.

The recognized DNA is *cut* in two, producing double-stranded sequences with sticky ends.

While most restriction enzymes — the class II restriction enzymes — recognize sequences that display a specific kind of symmetry (so called *palindromic* sequences) and cut at their recognition site, there exists a group of them — the class IIS restriction enzymes — that recognize nonpalindromic sequences and that cut at a distance from their recognition site.

These class IIS restriction enzymes can be used to implement a number of operations that have their use in simulating Turing machine configuration transitions: insertion, deletion and replacement of a DNA fragment (with orientation control), deletion and replacement without regenerating a restriction site and even the movement of a sequence through a strand of DNA (termed *progress*).

While similar operations are possible using class II restriction enzymes, there is a problem of control: most of them can have undesired side effects and are not specific enough.

Rothemund chose to use class IIS restriction enzymes, and the encoding for the instantaneous description and of the transition table is based for a large part on the property of class IIS restriction enzymes that the *cutting site* — the actual site where strands are cut — is determined by, but not equal to the restriction site.

4.3.2 Representing instantaneous descriptions

Remember that an instantaneous description of a Turing machine encodes the three variable components of the machine:

1. The contents of the tape
2. The position of the head
3. The state of the finite control

The contents of the tape. The contents of the tape form a string over the alphabet of the Turing machine. The individual symbols of the alphabet are each assigned an oligonucleotide sequence. Additionally, all symbols get the same short extra left and right sequences, that are equal for all symbols, and serve as markers for the begin and end of a symbol. The strings are encoded as the concatenation of the encodings of their constituent symbols.

The position of the head. The head is placed 'inside' of the tape. Because the head moves through the tape, there is a potential ambiguity: does the head point to the symbol to the right or to the left of it? While this could be resolved by adopting simple convention (e.g. the head always points to the symbol to the left), for implementation reasons, another resolution of the ambiguity is adopted: the head is made to consist of two distinct parts, one of which is the recognition site of a restriction enzyme; its splicing site is the current symbol. Thus, there are always two encodings for an instantaneous description, one for when the head has just performed a move to the left, and one for when the last move was to the right.

The state of the finite control. Lastly, the state of the finite control is encoded in the space between the recognition site of the part of the head pointing to the current symbol and that current symbol.

4.3.3 Transitions

Representing the transition table. The transition table is encoded into a number of oligonucleotides, which Rothemund appropriately terms *transition oligonucleotides*. They exist in four types: one for each combination of last move of the head (left or right) and next move.

The most important parts of the transition oligonucleotides are a sticky end (specific for each transition), a head sequence and a sequence for the new symbol.

Implementing the transitions. A transition is done in six steps that perform two processes: in the first, the head in a tape is replaced with the correct transition oligonucleotide; in the second, the previously read symbol is deleted.

Halting. A simple way of extracting results is to apply PCR amplification (with the halt sequence as a primer) after each step. When detected, the sequences containing the halt sequence can be sequenced, and the answer read out. A safer way is to bind to the halt sequence a group that can be used with bead extraction.

4.3.4 Estimates

Rothemund gives estimates of several aspects of the implementation of Minsky's smallest known universal Turing machine using his model.

Size. Representing the tremendous amount of one mole of bits (6×10^{23} bits) in appropriate solution concentrations requires only about 260 m^3 water. The individual tapes could encode some 80 kilobyte memory at maximum.

Speed. Transitions take about 4.5 hours (assuming reactions on solid support and enzymes at the recommended concentrations).

Energy. Transitions require some 44 KCal/mole DNA tapes.

4.3.5 Advantages

Rothemund's scheme is specified mostly in great detail.

It works entirely with double-stranded DNA, which is more stable than single-stranded DNA.

Reasonable estimates are given that it scales up to at least the scale of the smallest known universal Turing machine. This scale-up is in the size of the transition table (from the 'Busy Beaver' Turing machine that is Rothemund's running example).

4.3.6 Problems

There are some problems to Rothemund's scheme, some theoretical and some practical.

It does not describe how to generate the initial tapes. For the 'Busy Beaver' running example, the initial tape is blank; it is presumably relatively easy to sequence the strand encoding for instantaneous description of the Turing machine at the start of its computation, and then make sufficient copies of it (e.g. via PCR) to compensate for the practical problems discussed below. However, a universal Turing machine requires an initial tape with the encoding of the Turing machine to be simulated and its input; the initial strand can be much longer than that for the Busy Beaver case, and is therefore more difficult to generate.

The Busy Beaver example is a deterministic one. One can therefore indeed simply 'mix the twelve transition oligonucleotides [...] with the DNA Turing tapes'. Rothemund does not explain if his scheme is suitable for (or can be modified to suit) non-deterministic Turing machines: simply adding all transition nucleotides could result in interference in the reactions involving transition nucleotides that represent the different choices possible at the current time-step, while sequentially adding them (either in a strict order, or in random order) results in taking the same choice for all machines with the same status instead of having part of the machines take one choice, and the other machines take the other choices. It appears likely, though, that Beaver's suggestion to add random bits to the initial tape can be applied to Rothemund's model too; the 'fork using PCR' approach may work too, but is an additional source of errors.

The scheme requires unfortunately many different kinds of restriction enzymes, some of which may have poor performance or imperfect specificity of restriction. Failed restrictions can result in defective tapes. By incorporating suitable labels, these defective tapes can be removed directly after the step in which they have been generated, thus preventing their interference with subsequent steps. Incorrect restrictions can occur, but do so only very infrequently, when the restriction is performed under the recommended reaction circumstances.

The number of different kinds of restriction enzymes increases with the transition table size. Thus, Turing machines with large transition tables cannot be implemented with this scheme directly; one has to resort to simulation via a small universal Turing machine. The scheme can be used to implement the smallest known universal Turing machine, but simulations by this Turing machine are very inefficient. This is not a true problem, since Rothemund's goal was a proof of concept: universal computation is possible with Molecular Computation.

Also, the ligations involved may fail or even ligate mismatching pairs of sticky ends and ligation may occur between two tapes, instead of between parts of one tape. Suitable enzymes exist that can remove precisely the products of such undesired reactions.

Lastly, attachment of the DNA strands to solid support can be used to simplify the removal of reagents after the step in which they were needed is finished, and the tapes are kept separately, preventing them from ligating together.

4.4 Winfree's model: simulating cellular automata

Winfree has developed a model that is capable of universal computation based on a very different model capable of universal computation: cellular automata.

4.4.1 Cellular automata

Cellular automata ([CAF]) consist of a collection of cells (e.g. a two dimensional array), each containing a symbol from a finite alphabet, and a set of transition rules that are applied in parallel to all cells at fixed time intervals. The transition of a cell's content from one 'time tick' to another is determined by its current content and the contents of a finite number of 'neighbours' to it (e.g. for a two dimensional matrix, the cells directly left, right, above and below it).

A well-known example of a cellular automaton is Conway's game of life ([BCG82]). Cellular automata have a variety of applications, including the simulation of certain chemical reactions, and image processing. Also, they can be considered abstract models of spreadsheets.

There exist even one-dimensional cellular automata that are universal: for each Turing machine (including a universal one), one can construct a cellular automaton whose initial array contents correspond to the initial tape contents of the Turing machine, and whose transition rules simulate both the transition table, the cell replacement and the head movement of the Turing machine. A universal cellular automata thus has a fixed set of transition rules, and is programmed by providing the initial array contents.

There are a number of variations on the basic theme of cellular automata. One of these that is relevant to us is that of *blocked* cellular automata, a one-dimensional variation on partitioning cellular automata, which Winfree introduces. In this model, the transition rule is formulated for *pairs* of cells. There are two possible partitions, ways of dividing the cells into pairs of neighbours, in the array of a one-dimensional cellular automaton ($\ldots c_n c_{n+1} c_{n+2} c_{n+3} c_{n+4} \ldots$ can be paired like $\ldots (c_n c_{n+1})(c_{n+2} c_{n+3})(c_{n+4} \ldots$ or like $\ldots c_n)(c_{n+1} c_{n+2})(c_{n+3} c_{n+4}) \ldots$). During successive time steps the two possible partitions of cells in pairs are strictly alternated in the application of the transition rule.

Blocked cellular automata can be proven to be universal in a construction analogous to that for normal one-dimensional cellular automata. Winfree develops blocked cellular automata, since they can be simulated easily with the construction technique he is proposing.

4.4.2 Winfree's simulation

Winfree simulates a universal blocked cellular automaton by designing small units of DNA in such a way that they self-assemble into two-dimensional complexes according to the rules of the automaton in a hybridization reaction. In these complexes, a slice in one direction corresponds to the state of the whole automaton at a certain point in time, while a slice in the perpendicular direction shows the contents of one cell during the whole development of the automaton.

4.4.3 Evaluation

It is still unclear how practical Winfree's approach is. It depends on an unusual DNA structure, whose behaviour in practice has not been fully tested. [SWL$^+$96] discusses the large gap between theory and practice of constructing unusual DNA structures, including the tendency of DNA to form double-stranded helices, difficulty in control and the importance of studying the actual three-dimensional structure instead of relying on two-dimensional models. The problem in achieving substantial yield of desired results is less acute here, since the building blocks are simple enough to produce in sufficient quantities, and the complex structure forms as a result of the self-assembly of the building blocks.

Despite the dubious practicality, it has several features that may be inspirational to future research.

It is conceptually much simpler than Beaver's and Rothemund's models, using only one basic reaction (hybridization), in a straight-forward simulation, requiring no external processing (it is 'one-pot'). This illustrates the necessity of studying the many different models of computation for their suitability for implementation using molecular computational hardware, and of the search for a model of computation natural to molecular computation.

Also, it shows that the asynchronous nature of parallelism in biochemical reactions does not necessarily preclude approaches based on synchronous parallelism.

It may even be possible to use a similar self-assembling system to simulate Turing machines, although such a system would probably require many more, likely complex, building blocks, and would not use the parallelism that is natural to cellular automata.

Chapter 5

Splicing Systems

5.1 Background

While practical Molecular Computation started with Adleman's paper [Adl94], Formal Language theory already studied *splicing systems*, abstract models for the languages generated by strands of DNA under the application of restriction enzymes and subsequent annealing and ligation. These models were introduced by Head ([Hea87]).

A strong motivation of the study of DNA recombination using Formal Language theory is the complexity of several problems associated with DNA recombination such as shortest common superstring (encountered when determining the base sequence of a strand of DNA from the sequences of short substrands). This complexity is surprising given how simple the operators in DNA recombination appear to be. Studying these 'simple' operations capable of generating difficult problems may benefit complexity theory. As [Bea95d] put it: 'If DNA-related problems are difficult to solve, then DNA-based primitives may enable solutions to difficult problems'.

5.2 The splicing operator

The notation used here mostly follows [RS96, HU79]. For a detailed derivation of the concept of a splicing operator from DNA recombination, the reader is referred to [RS96].

Formal Language theory studies sets of strings (termed *languages*) over a finite *alphabet* V of *symbols* (like $\{0, 1\}$ and $\{A, C, T, G\}$). Strings (or *words*) are formed by applying the concatenation operation (notation '$\cdot$', but often implicit) to strings. The symbols from the alphabet are basic strings. E.g. $TAG \in V^*$ (for $V = \{A, C, T, G\}$; V^* is the language of strings over V) since $TAG = T \cdot (AG) = (TA) \cdot G = (T \cdot A) \cdot G$ (concatenation is associative) and T, A, G, are strings over V.

Splicing is the operation of concatenating a prefix of one string and a suffix of another string. E.g. applying splicing to strings 'snack' and 'tofu' may generate 'snafu'.

5.3 Splicing rules

This application of splicing, without rules to restrict its use, is, like similarly unrestricted concatenation, too general to be interesting (starting with only the symbols from the alphabet being considered, repeated splicing can generate all words over that alphabet), and is not very realistic (restriction enzymes are very specific about the recognition and splicing site).

Just as the use of concatenation is regulated by allowing it only on strings fulfilling requirements expressed in grammatical rules, the use of splicing is regulated by *splicing rules.*

A splicing rule is consists of four finite strings u_1, u_2, u_3, u_4 that are used as patterns. u_1, u_2 (u_3, u_4) determine the possible sites of the splicing in the first (second) string. u_1, u_4 are kept in the splicing result, while u_2, u_3 are not, assuming we only look at splicing with one result.

While it may be convenient to think of a splicing rule as consisting of four separate strings, for formal treatment it is easier to encode a splicing rule as a single string (so we can talk about the class of languages a set of splicing rules is in).

Formally, a *splicing rule* (over an alphabet V) is a string of the form

$$r = u_1\#u_2\$u_3\#u_4$$

where $\#$ and $\$$ are two special *marker* symbols not in V, and $u_i \in V^*(1 \leq i \leq 4)$.

For such a rule r, applying it to two strings x, y results in a string z

$(x, y, z \in V^*)$ as follows:

$$(x, y) \vdash_r z \text{ if and only if } \quad \begin{aligned} & x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2 \text{ and} \\ & z = x_1u_1u_4y_2,\ \text{for some } x_1, x_2, y_1, y_2 \in V^*. \end{aligned}$$

5.4 Splicing Systems

We can now proceed to the definition of an *H scheme*[1]. An H scheme is a pair

$$\sigma = (V, R) \quad \text{where} \quad \begin{aligned} & V \text{ is an alphabet} \\ & R \subseteq V^*\#V^*\$V^*\#V^* \text{ is a set of splicing rules} \end{aligned}$$

An H system $\sigma = (V, R)$ is used as a unary operator on languages. Applying σ once to a language $L \subseteq V^*$ yields

$$\sigma(L) = \{z \in V^* \mid (x, y) \vdash_r z,\ \text{for some } x, y \in L, r \in R\}$$

This can be used to study a single application of an H scheme. It can be extended to iterated application σ^* as follows:

$$\begin{aligned} \sigma^0(L) &= L, \\ \sigma^{i+1}(L) &= \sigma^i(L) \cup \sigma(\sigma^i(L)), i \geq 0, \\ \sigma^*(L) &= \textstyle\bigcup_{i \geq 0} \sigma^i(L) \end{aligned}$$

An H system, though normally viewed as an operator, can be likened to productions in the grammars of classical Formal Language theory. This similarity to grammars can be strengthened.

Classical grammars are ‘complete’ devices for generating languages. They specify an alphabet, a starting point, rules for combining generated strings into new ones, and terminal symbols out of which the output strings may consist. Associated with them is one interpretation: the language generated by the grammar.

Following this analogy closely, the concept of *extended H system* was introduced. An extended H system is a quadruple

$$\gamma = (V, T, A, R) \quad \begin{aligned} & V \text{ is an alphabet,} \\ & T \subseteq V \text{ is the } \textit{terminal alphabet}, \\ & A \subseteq V^* \text{ is the set of } \textit{axioms}, \\ & R \subseteq V^*\#V^*\$V^*\#V^* \text{ the set of splicing rules} \end{aligned}$$

[1] Following the precedent of L (Lindenmayer) systems, H schemes are named after their inventor, Thomas Head.

For such an extended H system $\gamma = (V, T, A, R)$, an *underlying H system* $\sigma = (V, R)$ is defined.

The *language generated by* γ is defined using σ as follows:

$$L(\gamma) = \sigma^*(A) \cap T^*$$

5.5 Classes of Splicing Systems

With traditional grammars, there is one 'knob' that can be turned in exploring their computational/generative power: the form of the productions.

Within extended H systems, there are two 'knobs': the classes of languages from which A, the set of axioms, and R, the set of splicing rules, are taken. E.g. [HPP96] shows that when both A and R are finite, extended H systems can produce regular languages; and that when A is kept finite, but R regular, the recursively enumerable languages can be produced (in other words: the full power of Turing machines).

5.6 Variations

While extended H systems are perhaps the most appealing model for a Formal Language theorist, because of their analogy to traditional grammars, there are several variants that are being studied.

Some of the more obvious are splicing systems in which an H system is applied only once, and splicing systems in which all symbols are terminal.

Splicing systems on multisets of strings, first studied in [DG89], are of practical interest, since they can accurately model the fact that strands are consumed in a splicing operation, as are splicing systems on circular strings, studied in e.g. [YKF95], which can model the behaviour of circular strands. Both of these variations can achieve universal computation for A and R from simple families in the Chomsky hierarchy.

Other variations restrict the applicability of splicing rules by allowing length-increasing results only, or on prefixes only etc., e.g. [KPS96].

Many variations have been proven to be capable of universal computation (e.g. [CVKP96, CVFKP96, DG89, FKP, Pău95, Pău96b, Pău96a, PRS96, YKF95]).

Lastly, there are generalisations of splicing to graphs and other non-string-like structures.

5.7 Interesting results

While splicing systems are interesting in themselves as abstract models in Formal Language theory, we are interested in them mostly for their original purpose: to model the languages of double strands of DNA generated under the influence of restriction enzymes and ligases. More precisely, are there splicing systems that can generate the recursively enumerable languages (i.e. are capable of universal computation), for which a realistic implementation is possible?

5.7.1 Requirements for practicality

There are a number of aspects in which splicing systems abstract away from practical biochemical limitations that become important again, when considering practical implementation.

First, in a practical model, the amount of initial strands and the number of different restriction enzymes is finite, so both the initial set and the set of axioms in a corresponding model will have to be finite.

Secondly, in practice, DNA strands are consumed in splicing: when strands w and z are generated from strands x and y, x and y are no longer available. This requirement is quite strict; we will not demand this in full: in most cases, the model still works when we assume a large, but finite, supply of all strands involved.

Thirdly, the length of a recognition site of a restriction enzyme is limited to 6–8 bases: restriction enzymes cannot recognize arbitrarily long sequences. This length forms the inspiration for the *radius* measure considered in [HPP96].

Lastly, some restrictions on the use of the splicing operator, like the *length-increasing* or *most-increasing* modes considered in [KPS96], are difficult to implement. We will not consider models based on such restrictions.

5.7.2 Candidate models for universal computation based on splicing

When we take into account the requirements just formulated, there are a few splicing system models that are practical, and capable of universal computation.

One is splicing systems based on multisets, as introduced in [DG89]. Incidentally, this is the first type of splicing system proven to be universal.

Another is that of splicing systems for circular strings, as studied in [YKF95].

Lastly, [HPP96] proves the existence of a universal (for a given alphabet) multiset splicing system with finite axioms and radius 2.

5.8 Problems

Splicing Systems at first sight appear to be an attractive model for developing practical Molecular Computation. However, there are as yet several severe problems that hinder their applicability in this way.

Only one type of chemistry. Splicing Systems were explicitly developed as models for DNA recombination. There are several other chemistries on which practical Molecular Computation might be based, like RNA editing, or the 'weird' DNA complexes used by Winfree. Focusing on Splicing Systems as *the* theoretical model for Molecular Computation would be voluntarily blinding oneself to the other possibilities, for some of which theoretical models to study their computational power may still have to be developed.

Unrealistic splicing. We have seen several barriers to directly implementing splicing systems. For example, restriction enzymes are capable of recognizing only rather short (6–8 base pairs) sequences, while splicing rules can recognize finite, but arbitrarily long, subwords.

Any potentially practical splicing system will have to use only a few restriction enzymes, since they are quite expensive, and function optimally under diverse reaction circumstances.

Finiteness. Practical systems will be finite. Incorporating this finiteness directly into splicing systems (by using multisets, a finite number of axioms and a finite number of splicing rules) easily results in models that are not capable of universal computation. Also, considering restrictions to enforce finiteness directly can be too restrictive, as a comparison with electronic computers illustrates.

Electronic computers are of course finite too, but we mostly consider them to be universal, but memorywisely challenged. The finiteness affects only one 'knob' of the Turing machine model: the tape size.

In splicing systems, the number of 'knobs' that can be tuned to produce a finite system is larger, and the 'knobs' are less well understood than memory.

The theory of Splicing Systems will have to be enhanced to gain insight into approximations that may produce practical models. For instance, if we have a universal splicing system with a small number of splicing rules, but which requires a regular set of axioms, an approximation would be to use a large but finite subset from these axioms, and see how much practical computing power is lost.

Chapter 6

Current developments in Molecular Computation

In this conclusive chapter, current developments in Molecular Computation are discussed, mainly on the basis of the papers presented at this and last year's DIMACS workshop on DNA Based Computers ([AMS96, BL96a]) and attendance reports of these meetings ([Smi95, Ame96b]).

6.1 Consideration of practical problems

6.1.1 Errors

No molecular computing scheme has yet been tested on a problem of more than toy size. In most schemes there are a number of sources of errors, e.g. the instability of single-stranded DNA in solution, hairpins and other undesired structures may form, ligation may occur between edges that are not fully Watson-Crick-complementary, templates may interact during PCR. In scaling up to useful problem sizes, these errors need to be dealt with.

There are several approaches to reduce errors or their effect on computations being studied.

Careful choice of encoding of information. A carefully chosen encoding can prevent the formation of undesired structures, such as hairpin loops, and can increase the difference (Hamming distance) between the coding of different elements, reducing the effect of binding between sequences that are not fully Watson-Crick-complementary. Examples of this approach are [Bau96b, DMG$^+$96, Mir96].

Other implementations of operations. For some operations, less error-prone implementations have been proposed. For instance, [AGH96] does not implement *extract* by magnetic bead extraction, but by destroying the undesired sequences (add complementary sequences to them, so that dsDNA is formed, then cut that with restriction enzyme).

Probabilistic/statistic approaches. When an operation is unreliable, its reliability can be increased by repeating it. For instance, if an *extract* operation is to be performed to separate strands which encode for a 1 in a certain position from those that encode for a 0 in that position, one can create a series of tubes in which each tube is labelled with the difference in the number of times the strands it contains have been classified as 0 from the number of times they were classified as 1. In this way, the strands perform a biased random walk between the tubes. This approach is taken in [KKW95] and in [RWB$^+$96], where it is termed a *refinery model.*

One can take advantage of these probabilistic effects in the encoding too. As [BDL96] shows, making the encoding redundant by simply encoding everything twice increases the reliability of *extract* implemented by magnetic beads extraction.

6.1.2 Attention for reaction circumstances

Some of the reactions used in implementing operations function optimally under reaction circumstances (temperature, pH etc.) that cause degradation of DNA. A balance needs to be struck between the necessity of performing these reactions fast and the acceptable degree of degradation of DNA.

6.2 Other information carriers or chemistries

Although linear single- or double-stranded DNA in solution is a suitable information carrier for molecular computation (it is well understood, fairly stable, and there is a large toolbox of operations to manipulate it), it is not the only suitable candidate.

Solid support / surface based approach. DNA need not be free in solution to be manipulated. It can also be attached to solid support on a surface. Solid support has some advantages: the strand loss that can occur in transport between tubes or vats can be eliminated, and by-products and enzymes can be washed out easily, leaving less possibilities for interference between subsequent steps. Solid support is considered in e.g. [Rot96, LGC^{+}96, CCC^{+}96].

Speedups. With a carefully designed chemistry, multiple different operations could be performed simultaneously (when they don't interfere with one another). For instance, the approach to addition of [GB96] is designed to allow for only one possible 'pipeline' in which strands undergo operations.

Similar effects, akin to pipelining in electronic architectures, could be achieved by employing selective membranes between vats, as in [RWB^{+}96]: it is no longer necessary to wait until an operation has been performed on all strands; as soon as a strand has undergone an operation, it can migrate to the vat in which the next operation is performed.

Non-linear DNA. As we have already seen in Section 4.4, non-linear DNA can be used to perform computations by suitable binding between 'building blocks'. It is however difficult to produce non-linear DNA in sufficient quantities and with sufficient control; these difficulties are discussed in [SWL^{+}96].

RNA. RNA shares much of the qualities of DNA, and may even be used as an enzyme to implement operations on itself. The biochemistry of RNA editing has possibilities for molecular computation, which are considered in [SS95, KMRS].

Artificial polymers. There are also artificial DNA-like polymers (e.g. PNA and DNG that use a different backbone material) that are more stable and have more specific binding than DNA. There are as yet no enzymes to manipulate them, so it is unlikely that they will replace DNA, but they may be useful in implementing operations (see [RWB^{+}96] for examples).

6.3 Hybrid schemes

Following [Lip94], there is interest in hybrid molecular/electronic computing schemes, i.e. schemes in which a molecular computing component functions as a 'subroutine' to perform massively parallel operations.

Trivial examples are the proposals in which an electronic computer or robot is used to automate the steps in a molecular computing scheme, like the 'parallel robotic workstation for molecular computation' of [RWB$^+$96].

6.4 Communication

Although molecular computers can achieve massive parallelism easily, in current schemes (except to a certain degree [Bea95d]), there are no real provisions for communication comparable to those in parallel electronic computers, such as semaphores and guarded expressions.

Communication can conceivably be used to 'recycle' strands that represent 'dead ends' in a search tree, or to implement the cycles in the evolutionary approach suggested in [Ste95] in one pool instead of in a series of pools.

Developing viable communication schemes is a challenge for several reasons. There is no addressing mechanism innate to strands in a solution[1], it has to be designed. Furthermore, communication will have to be asynchronous, since the chemical reactions are asynchronous, and will have to allow for a variable transport time for signals (signals encoded as molecules will have to float in solution until they hit their destination).

6.5 A 'killer application'?

It is as yet unclear if their is a 'killer application' for molecular computation — a single application or area of applications in which molecular computers are clearly superior to electronic ones, that is of such practical importance that it in itself is enough to stimulate and finance the further development of molecular computers.

[1] For surface-based molecular computing, there may be ones, but those are visible to the operator, not to the molecules performing the communication themselves.

While finding or developing a killer application would of course be of great importance to the field, it is debatable whether or not it is beneficial to the field to actively search for a killer application, and what the consequences of not finding one would be.

There have been computing architectures with what appeared to be killer applications, that still became dead ends in the evolution of computing, while other architectures had no killer application, but still survived, not because of technical superiority, but e.g. for economical reasons.

6.6 The future...

Practical molecular computation? As we have seen, Molecular Computation has great potential. Evaluating its practical potential is very difficult, because current implementations are for toy-size problems, not for real life ones. Although a number of problems associated with the scale-up which has to be realized for molecular computers to become practical tools, have been studied and approaches to overcoming them have been suggested, there may still be difficult problems ahead: in theory, there is no difference between theory and practice, but in practice...

The development of the field of Molecular Computation can be likened to Adleman's approach: a first phase 'generation of diversity', followed by a phase in which candidates that are shown to be unviable, are culled, hopefully resulting in a small number of practical models.

Currently, some models are being refined, while still some new ones are introduced using very different paradigms or implementations. As a consequence, one must be careful in interpreting results like [Har95b, Har95a, MD96], since they apply to one or more, but not necessarily all, models in Molecular Computation.

Theory. Regardless of whether or not molecular computing will be a viable technology, the study of Molecular Computation has provided us with a new way of viewing biological and chemical processes which may prove valuable in medicine and in understanding (the evolution of) life.

Also, it provides a stimulus to the study of Splicing Systems, which may deepen our understanding of the structure of language classes.

A warning. Currently, in Molecular Computation there is much more theory than experiment.

'*Beware of the Turing Tar-pit in which everything is possible but nothing of interest is easy.*' — `fortune(6)`

Appendix A

A bit of biochemical background

Many papers on molecular computation assume some knowledge about biochemistry, especially about DNA and RNA; we try to provide the basics of this background here by inspecting the roles they fulfill in nature. Part of the material in this section is based on [Res].

A.1 From DNA to proteins

A cell's genetic information is stored in strands of *DNA* (deoxyribonucleic acid). DNA is a polymer — a large molecule consisting of repeated smaller units (monomers) — with a linear structure. DNA consists of four different monomers known as *nucleotides.* All of these consist of a nitrogenous base (adenine (A), guanine (G), cytosine (C) or thymine (T)), a phosphate molecule and a sugar molecule. The phosphate and sugar molecules link together in a linear structure, thus forming DNA's *backbone.* The backbone has two distinct ends, known as 3' and 5' respectively, giving it a direction. This is also known as *polarity.* DNA can thus abstractly be viewed as a string over a four-letter alphabet, such as `TATAAGAGCAT`.

The genetic information in a cell must be passed through to a daughter-cell. The structure of DNA in a cell makes this *replication* easy. DNA in a cell is normally *double-stranded* (*dsDNA*): it forms a double helix of two intertwined strands that are bound together by hydrogen bounds. These hydrogen bonds occur with very strong preference between A and T and between C and G: the bases in these pairs are said to be Watson-Crick-complementary. The two strands are thus each other's complement: the base sequence of one is enough to determine the base sequence of the other. Conceptually DNA replication is achieved by splitting it into its two strands of *single-stranded* DNA (*ssDNA*)and building their complementary strand onto them.

Thus, strands `TATAAGAGCAT` and `ATATTCTCGTA` are Watson-Crick-complementary, and when they form Watson-Crick bounds, the resulting double strand is

```
TATAAGAGCAT
ATATTCTCGTA.
```

The genetic information that DNA encodes is the structure of proteins. Proteins are linear polymers that have a complex spatial structure, caused by sulphur bonds and by the hydrophilic and hydrophobic nature of the constituent amino acids. This structure is the reason for proteins' function as enzymes: biological chemicals that act as highly selective catalysts for the reactions that form a cell's metabolism. The monomers of proteins are amino acids. There are some twenty amino acids in nature; they are encoded by *codons*: three consecutive base-pairs.[1]

When a protein is constructed using the blueprint in a sequence of DNA, that sequence is said to be *expressed*. The expression of DNA is a complex process, involving various forms of *RNA* (ribonucleic acid), a chemical similar to DNA. RNA has a slightly different sugar in its backbone, making it more flexible, and uses uracil (U) instead of T; like T, U binds to A.

An enzyme *transcribes* (i.e. copies) the DNA sequence into a sequence of *messenger-RNA* (mRNA).

The mRNA sequence is read by *ribosomes*, parts of the cell that consist of *ribosomal RNA* (rRNA) and proteins. Based on the mRNA's instructions, a ribosome assembles the protein from amino-acids delivered to it by *transfer-RNA*. tRNA are small sequences of RNA that form an L-shaped spatial structure. A tRNA contains a site that binds to its specific amino-acid.

[1] A codon can encode $4^3 = 64$ possibilities. A few of these are start and stop instructions for the protein production process; the rest encode amino acids. The code is redundant and appears to have evolved in such a way as to reduce the effect of noise (read errors, mutations etc.) as much as possible ([Hof79]).

A.2 Manipulating DNA

There are several ways of manipulating DNA commonly used in Molecular Computation schemes.

A.2.1 Joining DNA sequences

dsDNA sequences can be joined when they begin or end in short overhanging complementary ssDNA sequences. These sequences are known as *sticky ends* or *cohesive ends*. The forming of hydrogen bonds between such sequences is known as *annealing* or *hybridization*. The resulting sequence has cuts in its backbone. These may be sealed using *DNA ligase*.

For example, $\begin{array}{l}\texttt{TATAAGA}\\ \texttt{ATATTCTCGTA}\end{array}$ can anneal to $\begin{array}{r}\texttt{GCATTAG}\\ \texttt{ATC}\end{array}$ to form $\begin{array}{l}\texttt{TATAAGAGCATTAG}\\ \texttt{ATATTCTCGTAATC}\end{array}$.

A.2.2 PCR

The *(DNA) polymerase chain reaction* (PCR) is a reaction in which double-stranded sequences of DNA — *templates* — are replicated using an enzyme from the class of DNA polymerases. PCR requires *primers*: short sequences from the start and the end of the sequences to be replicated.

Because the copies of templates can function as templates themselves, repeated PCR can be used to exponentially multiply the template.

PCR is often used to selectively amplify certain DNA sequences prior to a detection phase. It has to be used with care, since interactions between the templates themselves may result in 'weird' DNA, e.g. containing folds, that is difficult to distinguish from regular DNA using electrophoresis, currently the best DNA analysis procedure ([KCL96]).

A.2.3 Cutting DNA

Certain classes of enzymes, most notably *endonucleases*, are capable of cutting DNA at or near a *recognition sites*, a specific base sequence, producing most of the time strands with sticky ends. These enzymes are very useful for rewriting DNA, and in fact their function, combined with annealing and ligation, forms the inspiration for the splicing operator in Formal Language theory.

An example, from [Rot96], is *Fok*I which cuts

```
                  ↓
    GGATGNNNNNNNNNNNNNNN            GGATGNNNNNNNNN       NNNNNN
... CCTACNNNNNNNNNNNNNNN ... ⟶ ... CCTACNNNNNNNNNNNNN  +      NN ...
                      ↑
```

N is a wildcard for A,C,G,T. Note that the site of the actual cut is not equal to, but still determined by, the recognition site.

A.3 Candidate-molecules for universal Molecular Computation

For Adleman-style Molecular Computation molecules only have to represent candidate solutions. They are only information carriers, and are not rewritten. Techniques for reading them out and filtering them based on the information they encode are needed. DNA, RNA and possibly proteins are candidates for use in Adleman-style Molecular Computation.

Molecules that are not rewritten are not enough for universal computation. Universal computation essentially has two aspects: representation of state and iterated controlled state transformations. Transformation of state forms the main difficulty in Molecular Computation: the output of a partial computation must be reusable as input for further computation.

We have no tools to rewrite proteins, so they are unfit for universal Molecular Computation.

Studies of RNA have proven that it can function as an enzyme, and thus may be able to rewrite itself.[2] However, RNA might be too reactive, making control of state transformation very difficult.

Evolution has proven DNA to be a very good information carrier[3]. But, as far as we have seen, DNA is static in that role: it is expressed one-way. For rewriting DNA we can turn to mechanisms used by bacteria and viruses.

[2] This is essential to the *RNA world hypothesis* (e.g. [dD95]) in evolutionary microbiology: the hypothesis that there was a phase in the evolution of life in which RNA by itself performed all the functions of DNA, RNA and proteins/enzymes: carrying genetic information, expressing it, rewriting it and selectively catalyzing biochemical reactions.

[3] But not too good — mutation is important for evolution.

Bacteria employ enzymes that can be used for selectively rewriting DNA, especially restriction enzymes.

Bibliography

[Adl94] Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, November 11, 1994, http://www.hks.net/~cactus/doc/science/molecule_comp.html.

Abstract: The tools of molecular biology were used to solve an instance of the directed Hamiltonian path problem. A small graph was encoded in molecules of DNA, and the "operations" of the computation were performed with standard protocols and enzymes. This experiment demonstrates the feasibility of carrying out computations at the molecular level.

[Adl96] Leonard M. Adleman. On constructing a molecular computer. In Baum and Lipton [BL96a], ftp://usc.edu/pub/csinfo/papers/adleman/molecular_computer.ps. Based on Manuscript, Computer Science Department, University of Southern California, January 11, 1995.

Abstract: It has recently been suggested that under some circumstances computers based on molecular interactions may be a viable alternative to computers based on electronics. Here, some practical aspects of constructing a molecular computer are considered.

[AGH96] Martyn Amos, Alan Gibbons, and David Hodgson. Error-resistant implementation of DNA computations. In AMS96 [AMS96], http://www.csc.liv.ac.uk/~martyn/princeton.ps. Previously: Research Report CS-RR-298, Department of Computer Science, University of Warwick, Coventry CV4 7AL, England, January 1996.

Description: This paper introduces a new model of computation that employs the tools of molecular biology whose implementation

is far more error-resistant than extant proposals. We describe an abstraction of the model which lends itself to natural algorithmic description, particularly for problems in the complexity class **NP**. In addition we describe a number of linear-time algorithms within our model, particularly for **NP**-complete problems. We describe an *in vitro* realisation of the model and conclude with a discussion of future work.
A [Lip95a]-like model designed for error-resistance. "The main advantage of our model is that it doesn't repeatedly use the notoriously error-prone separation by DNA hybridization method to extract strands containing a certain subsequence." Instead, strands complementary to the undesired sequences are added, causing them to form dsDNA that can be cut with restriction enzymes with nearly 100effectiveness.

[Ame96a] John-Thones Amenyo. Mesoscopic computer engineering: Automating DNA-based molecular computing via traditional practices of parallel computer architecture design. In AMS96 [AMS96], `ftp://ftp.ans.net/pub/misc/DNAComparch.ps`.
Abstract: How does one go about automating the steps of DNA computing, or what amounts to the same thing, the practical engineering of *hands-free*, general-purpose DNA computers? The intent of this paper is to indicate how familiar computer design principles for electronic computers can be exploited to build practical computers at the mesoscopic scales of macromolecules and bio-polymers. DNA computing is the most realistic harbinger of such *molecular computers*. Pragmatically, it is expected that DNA computer architectures will be used routinely and not just for solving theoretically *hard* computational problems. The ideas discussed here are akin to the design of a practical programming language for a virtual computer. The paper shows that all proposed DNA computing algorithms can be run on parallel computer architectures configured from *trellis/lattice banks, filter banks* and *switching banks*. Thus, DNA computation can be re-interpreted as dataflow (or signal flow) networks and subject to conventional treatment.

[Ame96b] John-Thones Amenyo. Workshop report: Personal impressions about the 2nd Annual Workshop on DNA Computing, June 21, 1996, `ftp://ftp.ans.net/pub/misc/jta/DNAComp2rept.txt`.

[AMS96] American Mathematical Society. *Proceedings of the Second Annual Meeting on DNA Based Computers, held at Princeton University, June 10-12, 1996.*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science., ISSN 1052-1798, 1996. To appear.
Contains [ARRW96], [Ame96a], [AGH96], [BB96], [BL96b], [BDLS96], [DMG$^+$96], [GB96], [JK96], [KCL96], [KMRS96], [LSW$^+$96], [LGC$^+$96], [Mir96], [Oli96], [Pău96a], [RWB$^+$96], [SWL$^+$96], [WW96], [WYS96]. Program committee: Eric Baum, Dan Boneh, Peter Kaplan, Richard Lipton, John Reif and Nadrian Seeman.

[ARRW96] Leonard M. Adleman, Paul W. K. Rothemund, Sam Roweis, and Erik Winfree. On applying molecular computation to the data encryption standard. In AMS96 [AMS96], `ftp://hope.caltech.edu/pub/pwkr/DIMACS/des.ps`.
Abstract: Recently, Boneh, Dunworth, and Lipton described the potential use of molecular computation in attacking the United States Data Encryption Standard (DES). Here, we provide a description of such an attack using the *sticker model* of molecular computation. Our analysis suggests that such an attack might be mounted on a table-top machine, using approximately a gram of DNA and might succeed even in the presence of a large number of errors.

[Bau95] Eric B. Baum. Building an associative memory vastly larger than the brain. *Science*, 268:583–585, April 28, 1995.
Abstract The techniques of [Adl94] and [Lip94] may be usable to construct an associative (= content-addressable) memory with a capacity that exceeds that of the human brain. Given a part of the content, this part can be used in extracting those molecules that match it. The "cue" can be used e.g. with magnetic bead extraction.

[Bau96a] Eric B. Baum. Building an associative memory vastly larger than the brain. In Baum and Lipton [BL96a].
The techniques of [Adl94] and [Lip94] may be usable to construct an associative (= content-addressable) memory with a capacity that exceeds that of the human brain. Given a part of the content, this part can be used in extracting those molecules that match it. The "cue" can be used e.g. with magnetic bead extraction.

[Bau96b] Eric B. Baum. DNA sequences useful for computation, `http://www.neci.nj.nec.com/homepages/eric/seq.ps`. June 1996.
Abstract: Recent proposals for DNA based computing [Adl94], [Lip95a], [Bau95] encode Boolean vector component values with sequences of DNA. It has previously been assumed that sufficient length random subsequences could be used to encode component values. However use of such subsequences will inadvertently result in long complementary subsequences. Complementary subsequences of sufficient length would stick to each other and cause mistakes or delays in computation. We suggest some constraints on DNA subsequences to be used in encodings, and describe maximal sets of subsequences satisfying these constraints.
[Adl94, Lip94] work with codes based on random subsequences of DNA as codewords. In strings in these codes, there may be long complementary subsequences that can result in undesired annealing. Furthermore, the codewords might even self-anneal. In this paper, the problem of finding a code that does not suffer from these problems is formalized and solved. A similar code is termed *unequivocable* in [Bea95d].

[BB96] Eric B. Baum and Dan Boneh. Running dynamic programming algorithms on a DNA computer. In AMS96 [AMS96], `http://www.neci.nj.nec.com/homepages/eric/dpr.ps`.
Abstract: In this paper we show that DNA computers are especially useful for running algorithms which are based on dynamic programming. This class of algorithms takes advantage of the large memory capacity of a DNA computer. We present algorithms for solving certain instances of the knapsack problem using a dynamic programming approach. Unlike other algorithms [Adl94], [Lip95a] for DNA computers, which are brute force, dynamic programming is the same algorithm one would use to solve (smaller) problems on a conventional computer.

[BCG82] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for your mathematical plays*, volume 2: games in particular. Academic Press (Harcourt Brace Jovanovich), third printing 1985 with corrections edition, 1982.

[BDL95] Dan Boneh, Christopher Dunworth, and Richard J. Lipton. Breaking DES using a molecular computer. Technical Report

CS-TR-489-95, Princeton University, May 1995, `ftp://ftp.cs.princeton.edu/pub/people/dabo/bioDES.ps.Z`. To appear in IEEE COMPUTER;

Description: Recently Adleman has shown that a small traveling salesman problem can be solved by molecular operations. In this paper we show how the same principles can be applied to breaking the Data Encryption Standard (DES). Our method is based on an encoding technique presented by Lipton. We describe in detail a library of operations which are useful when working with a molecular computer. We estimate that given one arbitrary (plain-text, cipher-text) pair, one can recover the DES key in about 4 months of work. Furthermore, if one is given cipher-text, but the plain text is only known to be one of several candidates then it is still possible to recover the key in about 4 months of work. Finally, under chosen cipher-text attack it is possible to recover the DES key in one day using some preprocessing.

[BDL96] Dan Boneh, Christopher Dunworth, and Richard J. Lipton. Breaking DES using a molecular computer. In Baum and Lipton [BL96a].

See [BDL95]

[BDLS96] Dan Boneh, Christopher Dunworth, Richard J. Lipton, and Jiří Sgall. Making DNA computers error resistant. In AMS96 [AMS96].

Abstract: We describe methods for making volume decreasing algorithms more resistant to certain types of errors. Such error recovery techniques are crucial if DNA computers ever become practical. Our first approach relies on applying PCR at various stages of the computation. We analyze its performance and show that it increases the survival-probability of various strands to acceptable proportions. Our second approach relies on changing the method by which information is encoded on DNA strands. This encoding is likely to reduce false negative errors during the bead separation procedure.

Introduces two methods to deal with common sources of errors:

- PCR after each selection step is shown to increase good strands' survival probability (for volume-decreasing salgorithms uch as Adleman's or Lipton's).
- Double encoding improves the probability that correct strands are extracted.

Note that [KCL96, Smi95] suggest PCR may be a major source of errors itself!

[BDS96] Dan Boneh, Christopher Dunworth, and Jiří Sgall. On the computational power of DNA. *Discrete Applied Mathematics*, 71(1-3):79–94, 1996. Also Technical Report, TR-499-95, Princeton University, october 1995. `ftp://ftp.cs.princeton.edu/pub/people/dabo/biocircuit.ps.Z`, `ftp://ftp.cs.princeton.edu/reports/1995/499.ps.Z`.
Abstract We show how DNA-based computers can be used to solve the satisfiability problem for boolean circuits. Furthermore, we show how DNA computers can solve optimization problems directly wit problems. Our methods also enable random sampling of satisfying assignments.

[Bea94] Donald Beaver. Factoring: The DNA solution. In Josef Pieprzyk and Reihanah Safavi-Naini, editors, *Advances in Cryptology - Asiacrypt '94 Proceedings 4th International Conference on the Theory and Applications of Cryptology.*, number 917 in Lecture Notes in Computer Science, pages 419–423, Wollongong, Australia, November–December 1994. Springer Verlag, Berlin, Heidelberg, New York., ISBN 3-540-59339-X, `http://www.transarc.com/afs/transarc.com/public/beaver/html/research/al%ternative/molecute/publications/b94asia.ps`. Extended abstract. The full version is [Bea95a].
Summary How to factor and compute NP functions using DNA, using a novel procedure for site-directed mutagenesis.
Abstract We consider molecular models for computing and derive a DNA-based mechanism for solving intractable problems through massive parallelism. In principle, such methods might reduce the effort needed to solve otherwise difficult tasks, such as factoring

large numbers. We investigate the application of such techniques to cryptography.

[Bea95a] Donald Beaver. Computing with DNA. *Journal of Computational Biology*, 2(1):1–8, Spring 1995, `http://www.transarc.com/afs/transarc.com/public/beaver/html/research/al%ternative/molecute/publications/bc.ps`. Full version of [Bea94].
Summary How to factor and compute NP functions using DNA, using a novel procedure for site-directed mutagenesis.
Abstract We consider molecular models for computing and derive a DNA-based mechanism for solving intractable problems through massive parallelism. In principle, such methods might reduce the effort needed to solve otherwise difficult tasks, such as factoring large numbers, a computationally-intensive task whose intractability forms the basis for much of modern cryptography.

[Bea95b] Donald Beaver. Molecular computing. Technical Report TR95-001, Penn State University, January 31 1995, `http://www.transarc.com/afs/transarc.com/public/beaver/html/research/al%ternative/molecute/publications/TR95-001.ps`.
Summary How to build and operate a Turing machine consisting of a single DNA molecule. How to compute NP and PSPACE functions using massively parallelized molecular computations. (Refinements, such as a more efficient encoding, or simplified experimental techniques, are not included.)
Abstract We design a molecular Turing machine and determine the complexity of the problems solvable by molecular computers. In [Adl94], a combinatorial molecular experiment to solve the NP-complete problem of Hamiltonian Path was proposed and implemented. Using our design, we show that such molecular computers can in fact compute PSPACE, under the generous assumptions implicit in [Adl94]. Under stronger and somewhat more practical restrictions, which [Adl94] fails to satisfy, we show that molecular computers are limited to solving problems in P.

[Bea95c] Donald Beaver. A universal molecular computer, 1995, `http://www.transarc.com/afs/transarc.com/`

public/beaver/html/research/al%ternative/molecute/publications/dimacs95.ps. Condensed abstract (of [Bea95b]) for DIMACS Workshop of April 4, 1995.

Summary How to build and operate a Turing machine consisting of a single DNA molecule. How to compute NP and PSPACE functions using massively parallelized molecular computations. (Refinements, such as a more efficient encoding, or simplified experimental techniques, are not included.)

Abstract We design a molecular Turing machine and determine the complexity of the problems solvable by molecular computers. Interest in "nanocomputation" has been sparked by Adleman's recent experiment demonstrating the possibility that molecular computers might solve intractable problems, such as Hamiltonian Path, using large-scale parallelism achievable only through molecular-scale miniaturization. We propose a method for site-directed mutagenesis (namely, a molecular "editing" reaction) and use it to build a universal computer, stepping beyond Adleman's special-purpose, one-time problem solver. Using the generous assumptions on parallelism implicit in Adleman's methods, we show that molecular computers can in fact compute PSPACE. Under stronger and more realistic restrictions, we show that molecular computers — both ours and Adleman's — are limited to solving problems in BPP.

[Bea95d] Donald Beaver. Universality and complexity of molecular computation, http://www.transarc.com/afs/transarc.com/public/beaver/html/research/al%ternative/molecute/publications/psp95.ps. Extended abstract. Submitted to Twenty-eighth Annual ACM Symposium on Theory of Computing 1996 (STOC)., 1995.

Abstract Adleman recently designed and executed an experiment to solve instances of the Hamiltonian Path problem using DNA molecules ([Adl94]). Two questions naturally arise, both of which we answer in this paper: First, is universal computation possible? Second, does NP characterize the limit of such computation? We design a (nondeterministic) Turing machine based on interactions of small DNA molecules, supporting general-purpose computation rather than just special-purpose oracle queries. Our construction supports massively parallel, synchronized operations of heterogeneous, communicating, nondeterministic Turing machines, using fairly conventional techniques from molecular biology. In the loosely restricted model implicit in Adleman's solution to Hamiltonian Path, we show that molecular computation is capable not merely of NP but of PSPACE. More generally, our results show how to utilize the parallelism of molecular computation to conduct any $S(n)$-space-bounded computation in $O(S(n))$ laboratory steps using molecules of size $O(S(n))$.

[Bea96] Donald Beaver. A universal molecular computer. In Baum and Lipton [BL96a]. See [Bea95c].
Description: This volume presents the proceedings of a conference held at Princeton University on April 4, 1995 as part of the DIMACS Special Year on Mathematical Support for Molecular Biology. The subject of the conference was the new area of DNA based computing. DNA based computing is the study of using DNA strands as individual computers. The concept was initiated by Leonard Adleman's paper in Science in November 1994. Contains [Adl96], [Bau96a], [Bea96], [BDL96], [Lip96], [Rot96], [SS96], [Win95a], [Win95b].

[BL96a] Eric B. Baum and Richard J. Lipton, editors. *DNA Based Computers*, volume 27 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science.* ISSN 1052-1798. American Mathematical Society, 1996, ISBN 0-8218-0518-5. Also known under the working title *DNA Computing.*
Description: This volume presents the proceedings of a conference held at Princeton University on April 4, 1995 as part of the DIMACS Special Year on Mathematical Support for Molecular Biology. The subject of the conference was the new area of DNA based computing. DNA based computing is the study of using DNA strands as individual computers. The concept was initiated by Leonard Adleman's paper in Science in November 1994. Contains [Adl96], [Bau96a], [Bea96], [BDL96], [Lip96], [Rot96], [SS96], [Win95a], [Win95b].

[BL96b] Dan Boneh and Richard Lipton. Making DNA computers error resistant. In AMS96 [AMS96], `ftp://ftp.cs.princeton.edu/pub/people/dabo/bioerror.ps.Z`, `ftp://ftp.cs.princeton.edu/reports/1995/491.ps.Z`. Previously: Princeton University TR-491-95, May 1995

Description: Recently Lipton showed that the formula satisfaction problem can be solved using a DNA based computer. The algorithm ignored the effects of errors that occur during biological experiments. In this paper we show that Lipton's algorithm can be made resistant to errors. In addition, we present a new circuit satisfaction algorithm which can be made error resistant using the same techniques.

[CAF] Frequently Asked Questions about Cellular Automata: Contributions from the CA community edited by Howard Gutowitz. `http://alife.santafe.edu/alife/topics/cas/ca-faq/ca-faq.html`, `ftp://alife.santafe.edu/pub/topics/cas/postscript/`, `ftp://alife.santafe.edu/pub/topics/cas/txt/`.

[CCC+96] Weiping Cai, Anne E. Condon, Robert M. Corn, Elton Glaser, Zhengdong Fei, Tony Frutos, Zhen Guo, Max G. Lagally, Qinghua Liu, Lloyd M. Smith, and Andrew Thiel. The power of surface-based dna computation, July 1 1996, `ftp://corninfo.chem.wisc.edu/Papers/powerDNA.ps`. Preprint.

Abstract A new model of DNA computation that is based on surface chemistry is studied. Such computations involve the manipulation of DNA strands that are immobilized on a surface, rather than in solution as in the work of Adleman. Surface-based chemistry has been a critical technology in many recent advances in biochemistry and offers several advantages over solution-based chemistry, including simplified handling of samples and elimination of loss of strands, which reduce error in the computation. The main contribution of this paper is in showing that surface-based DNA chemistry efficiently supports general circuit computation on many inputs in parallel. To do this, an abstract model of computation that allows parallel manipulation of binary inputs is described. It is then shown that this model can be implemented using fairly standard chemistry, in which inputs are encoded as DNA strands and the strands are repeatedly modified in parallel on a surface using the chemical processes of hybridization, exonuclease degradation, polymerase extension or ligation. Thirdly, it is shown that the model supports efficient circuit simulation in the following sense: exactly those inputs that satisfy a circuit can be isolated, and the number of parallel operations needed

to do this is proportional to the size of the circuit. Finally, results are presented on the power of the model when another resource of DNA computation is limited, namely strand length.
Presents an abstract model of the surfaced-based approach to DNA computation presented in [LGC$^+$96] and describes its power and limitations.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Electrical Engineering and Computer Science Series. The MIT Press, 1990.

[CVFKP96] 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, ISBN 981-02-2578-4, `http://www.cgl.ucsf.edu/psb/psb96/proceedings/cshhaj-varju.ps`. Also: Technical report 185-2/FR-2/95, TU Wien, Institute for Computer Languages, Wien, Austria, 1995, `http://www.csd.uwo.ca/~lila/four.ps`.
Abstract The paper extends some of the most recently obtained results on the computational universality of extended H systems (with regular sets of rules respectively with finite sets of rules used with simple additional mechanisms) and shows the possibility to obtain universal systems based on these extended H systems, i.e. the theoretical possibility to design programmable universal DNA computers based on the splicing operation. The additional mechanisms considered here are: multisets (counting the numbers of copies of each available string), checking the presence/absence of certain symbols in the spliced strings, and organizing the work of the system in a distributed way (like in a parallel communicating grammar system). In the case of multisets we also consider the way of simulating a Turing machine (computing a partial recursive function) by an equivalent H system (computing the same function), in the other cases we consider the interpretation of algorithms as language generating devices, hence the aim is to reach the power of Chomsky type-0 grammars, the standard model for representing algorithms equivalent with Turing machines taken as language generators.

[CVKP96] Erzsébet Csuhaj-Varjú, Lila Kari, and Gheorghe Păun. Test tube distributed systems based on splicing. *Computers and AI*, 15(2–3):211–232, 1996, `http://www.csd.uwo.ca/~lila/dnapcgs.ps`.

Abstract We define a symbol processing mechanism with the components (test tubes) working as splicing schemes in the sense of T. Head and communicating by redistributing the contents of tubes (in a similar way to the *separate* operation of Lipton-Adleman). (These systems are similar to the distributed generative mechanisms called Parallel Communicating Grammar Systems.) Systems with finite initial contents of tubes and finite sets of splicing rules associated to each component are computationally complete, they characterize the family of recursively enumerable languages. The existence of universal test tube distributed systems is obtained on this basis, hence the theoretical proof of the possibility to design universal programmable computers with the structure of such a system.

[Das] J.H.M. Dassen. A bibliography of molecular computation and splicing systems. HTML: `http://www.wi.LeidenUniv.nl/~jdassen/dna.html`, BibTeX source: `http://www.wi.LeidenUniv.nl/~jdassen/dna.bib`. This bibliography is hooked into `http://liinwww.ira.uka.de/bibliography/index.html`, The Collection of Computer Science Bibliographies.

Description A hyperbibliography on the subject of Molecular Computation and the related theoretical model of Splicing Systems. Molecular Computation is computation using (biological) macromolecules like DNA as information carriers, that are manipulated using biological operators, such as enzymes, and operations commonly used in bio-technology and genetic manipulation, such as filtering operations and the polymerase chain reaction. It has received much attention following Adleman's seminal article [Adl94]. Splicing Systems are models in Formal Language Theory that use the splicing operator instead of concatenation. The splicing operator is an operator on two strings that is an abstraction of the effect of restriction enzymes on strands of double-stranded DNA combined with ligation. It was introduced in [Hea87].

[dD95] Christian de Duve. The beginnings of life on Earth. *American Scientist*, 83(5), September-October 1995, `http://www.sigmaxi.org/amsci/articles/95articles/CdeDuve.html`.
A discussion of the RNA world and a possible pre-RNA world.

[DG89] K.L. Denninghoff and R.W. Gatterdam. On the undecidability of splicing systems. *International Journal of Computer Mathematics*, 27:133–145, 1989.
Abstract: The notion of *splicing system* has been used to abstract the process of DNA digestion by restriction enzymes and subsequent religation. A *splicing system language* is the formal language of all DNA strings producible by such a process. The *membership problem* is to devise an algorithm (if possible) to answer the question of whether or not a given DNA string belongs to a splicing system language given by initial strings and enzymes. In this paper the concept of a *sequential splicing system* is introduced. A sequential splicing system differs from a splicing system in that the latter allows arbitrarily many copes of any string in the initial set whereas the sequential splicing system may restrict the initial number of copies of some strings. The main result is that there exist sequential splicing systems with recursively unsolvable membership problem. The technique of the proof is to embed Turing machine computations in the languages.
Introduces multisets into splicing systems, and shows such splicing systems to be universal.

[DMG$^+$96] R. Deaton, R.C. Murphy, M. Garzon, D.R. Franceschetti, and S.E. Stevens, Jr. Good encodings for DNA-based solutions to combinatorial problems. In AMS96 [AMS96], `http://www.ee.memphis.edu/~rdeaton/pubs/dna_codes.ps`.
Abstract: Adleman has solved the Hamiltonian path problem by encoding the vertices and edges of the graph in oligonucleotides of DNA, hybridizing the oligonucleotides to produce potential answers, and extracting any DNA which corresponds to the Hamiltonian path. Depending on the conditions under which the DNA reactions occur, two oligonucleotides can hybridize without exact matching between their base pairs. This possibility was verified by experiment. For DNA-based solutions to combinatorial problems to become a viable

and practical technology, the possibility of false positives must be eliminated. The primary mechanism for the production of false positives is hybridization stringency that depends on the reaction conditions, of which the most important is temperature. Evidence is provided that encoding the vertices and edges of the graph in DNA oligonucleotides that are a minimum distance apart results in reliable encodings that virtually eliminate the risk of false positives. A genetic algorithm was shown to be useful to search the space of possible codewords. The Hamming bound is shown to be an upper bound on the number of reliable encodings. Laboratory results confirmed that the choice of good encodings is very dependent on the reaction conditions.

[FKP] Rudolf Freund, Lila Kari, and Gheorghe Păun. DNA computation based on splicing: The existence of universal computers. *Journal of the ACM*, `http://www.csd.uwo.ca/~lila/jacm.ps`. To appear. Also Technical Report 185-2/FR-2/95, TU Wien, 1995.

Abstract Splicing systems are generative mechanism based on the splicing operation introduced by Tom Head as a model of DNA recombination. We prove that the generative power of finite extended splicing systems equals that of Turing machines, provided we consider multisets or provided a control mechanism is added. We also show that there exist universal splicing systems with the properties above, i.e. there exists a universal splicing system with fixed components which can simulate the behaviour of any given splicing system, when an encoding of the particular splicing system is added to its set of axioms. In this way the possibility of designing programmable DNA computers based on the splicing operations is proved.

[GB96] Frank Guarnieri and Carter Bancroft. Use of a horizontal chain reaction for DNA-based addition. In AMS96 [AMS96].

[Gif94] David K. Gifford. On the path to computation with DNA. *Science*, 266:993–994, November 11, 1994, `http://www.hks.net/~cactus/doc/science/molecule_comp_perspect.html`.

An essay on molecular computation in the 'perspective' section of *Science*, in the same issue as [Adl94]. It discusses the promises of molecular computation

- "DNA ligation can effectively search a large space of potential solutions", and similar techniques may be developed for molecule design (e.g. for proteins).
- unheard of information representation density
- extremely energy-efficient

and the problems

- not practical enough yet

"There may be other computational processes lurking behind seemingly simple biological processes".

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, San Francisco, 1979.

[Har95a] Juris Hartmanis. On the computing paradigm and computational complexity. In Jiří Wiederman and Petr Hájek, editors, *Mathematical Foundations of Computer Science 1995. 20th International Symposium, MFCS '95. Proceedings.*, volume 969 of *Lecture Notes in Computer Science*, pages 82–92, Prague, Czech Republic, August-September 1995. Springer Verlag, Berlin, Heidelberg, New York., ISBN 3-540-60246-1.

Abstract: Computational complexity theory is the study of the quantitative laws that govern computing. Since the computing paradigm is universal and pervasive, the quantitative laws of computational complexity apply to all information processing from numerical computations and simulation to logical reasoning and formal theorem proving, as well as processes of rational reasoning. In this view, the search for what is and is not feasibly computable takes on an even deeper significance than just a central problem in theoretical computer science. The search for the limits of what is feasibly computable is the search for the limits of scientific theories and, possibly, rational reasoning.

An overview of the state of computational complexity theory. Reiterates the argument of [Har95b] that Molecular Computation cannot break the exponential barrier.

[Har95b] Juris Hartmanis. On the weight of computations. *Bulletin of the European Association for Theoretical Computer Science*, 55:136–138, February 1995.

Shows that Molecular Computation cannot break the exponential barrier: exponential-complexity algorithms remain infeasible even for fairly small problem instances. Applying the approach of [Adl94] to a 200-node graph would require an amount of DNA weighing more than the Earth. The main argument is reiterated in [Har95a].

[Hea87] Thomas Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49(6):737–759, 1987.

Abstract A new manner of relating formal language theory to the study of informational macromolecules is initiated. A language is associated with each pair of sets where the first set consists of double-stranded DNA molecules and the second set consists of the recombinational behaviors allowed by specified classes of enzymatic activities. The assosciated language consists of strings of symbols that represent the primary structures of the DNA molecules that may potentially arise from the original set of DNA molecules under the given enzymatic activities. Attention is focused on the potential effect of sets of restriction enzymes and a ligase that allow DNA molecules to be cleaved and reassociated to produce further molecules. The associated languages are analaysed by means of a new generative formalism called a splicing system. A significant subclass of these languages, which we call the persistent splicing languages, is shown to coincide with a class of regular languages which have been previously studied in other context: the strictly locally testable languages. This study initiates the formal analysis of the generative power of recombinational behaviors in general. The splicing system formalism allows observations to be made concerning the generative power of general recombination and also of sets of enzymatic activities that include general recombination.

[Hof79] Douglas R. Hofstadter. *Gödel, Escher, Bach: an eternal golden braid.* Basic Books, 1979, ISBN 0-394-74502-7.

An inspired book on strange loops, the nature of intelligence, Gödel's theorem, computability and many other themes. It contains a very clear description of the process of protein synthesis, the levels of description involved and the "location" of the genetic code.

[HPP96] Thomas Head, Gheorghe Păun, and Dennis Pixton. *Generative Mechanisms Suggested by DNA Recombination.* Volume 2 of Rozenberg and Salomaa [RS96], October 1996, ISBN Vol 1: 3-540-60420-0, Vol 2: 3-540-60648-3, Vol 3: 3-540-60649-1.
Description
Volume 1: Word, Language, Grammar
This first volume of the Handbook of Formal Languages gives a comprehensive authoritative exposition on the core of language theory. Grammars, codes, power series, L systems, and combinatorics on words are all discussed in a thorough, yet self-contained manner. This is perhaps the most informative single volume in the history of theoretical computer science.
Volume 2: Linear Modeling: Background and Application
This second volume of the Handbook of Formal Languages contains the most fundamental applications of language theory. Various aspects of linguistics and parsing, both natural and programming languages, symbolic manipulation, and pattern matching are discussed. A special feature is the recently very active field of DNA computing.
Volume 3: Beyond Words
This third volume of the Handbook of Formal Languages discusses language theory beyond linear or string models: trees, graphs, grids, pictures, computer graphics. Many chapters offer an authoritative self-contained exposition of an entire area. Special emphasis is on interconnections with logic.
Volume 2 contains [HPP96].

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[JK96] Nataša Jonoska and Stephen A. Karl. A molecular computation of the road coloring problem. In AMS96 [AMS96].
Abstract: Two algorithms for molecular computation of the road coloring problem are presented. We present in detail the laboratory techniques to implement these algorithms. In both of these algorithms a new operation of substring matching in the process of sep-

arating molecules is introduced. The laboratory techniques of the implementation are discussed.

[KCL96] Peter D. Kaplan, Guillermo Cecchi, and Albert Libchaber. DNA based molecular computation: template-template interactions in PCR. In AMS96 [AMS96].

Abstract: Since Adleman performed a computation with molecules of DNA [Adl94], there has been theoretical work on parallel computing with DNA [Lip95a], [Rei95], [Bea95a], [BDL95] but no experimental review of the promises and complications of DNA based computing. In this experiment, we focus on heteroduplex formation during the polymerase chain reaction as one critical complication to large scale DNA computing.

- Analysis of Adleman's method on even simpler graphs.
- Shows PCR (used in 'extract' and 'amplify') to be a source of errors due to template-template interactions (dsDNA is separated into single strands, that are intended to bind to the primers, but also bind to each other) resulting in "weird" DNA (e.g. with folds).
- Electrophoresis cannot distinguish normal and "weird" DNA well enough, but no better method for analysing DNA currently exists.
- High concentrations of template or product amplify template-template interaction problems.

[KKW95] Richard Karp, Claire Kenyon, and Orli Waarts. Error resilient DNA computation. Research report 95-20, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, 46, Allée d'Italie 69364 LYON CEDEX 07 - FRANCE, September 1995, `ftp://ftp.lip.ens-lyon.fr/pub/Rapports/RR/RR95/RR95-20.ps.Z`.

Abstract The DNA model of computation, with test tubes of DNA molecules encoding bit sequences, is based on three primitives, extract-a-bit, merge-two-tubes and detect-emptiness. Perfect operations can test the satisfiability of any boolean formula in linear time. However, in reality the extract operation is faulty. We determine the minimum number of faulty extract operations required to simulate a single highly reliable extract operation, and derive a method for converting any algorithm based on error-free operations to an error-

resilient one.
Achieves a more reliable *extract* by repeatedly performing *extract* in a series of test tubes. The DNA strands perform a biased random walk between the tubes.

[KMRS] Stuart A. Kurtz, Stephen Mahaney, James Royer, and Janos Simon. Biological computing, `http://www.cs.uchicago.edu/~stuart/Research/bc.ps`. In L. Hemaspaandra and A. Selman, editors, *Complexity Retrospective II*. To appear.
Abstract: Adleman's [Adl94] successful solution of a seven-vertex instance of the NP-complete Hamiltonian Path problem by recombinant DNA technology initiated the field of biological computing. We propose a very different model of molecular computing based on the biochemistry of RNA editing and RNA translation. In our model, individual molecules become fully capable general purpose computers.

[KMRS96] Stuart A. Kurtz, Stephen R. Mahaney, James S. Royer, and Janos Simon. Active transport in biological computing (preliminary version). In AMS96 [AMS96], `http://www.cs.uchicago.edu/~stuart/Research/transport.ps`.
Abstract: Early papers on biological computing focussed on combinatorial and algorithmic issues, and worked with intentionally oversimplified chemical models. In this paper, we reintroduce complexity to the chemical model by considering the effect problem size has on the initial concentrations of reactants, and the effect this has in turn on the rate of production and quantity of final reaction products. We give a sobering preliminary analysis of Adleman's technique for solving Hamiltonian path. Even on the simplest problems, the annealing phase of Adleman's technique requires time $\Omega(n^2)$ rather than the $O(\log n)$ complexity given by a computationally inspired but chemically naive analysis. On more difficult problems, not only does the rate of production of witnessing molecules drop exponentially in problems size, the final yield also drops exponentially. These issues are not objections to biological computing *per se*, but rather difficulties to be overcome in its development as a viable technology.

[KPS96] Lila Kari, Gheorghe Păun, and Arto Salomaa. The power of restricted splicing with rules from a regular language. *The Journal* *of Universal Computer Science*, 2(4):224–240, April 1996, `http://hyperg.iicm.tu-graz.ac.at/the_power_of_restricted_splicing_with_%rules_from_a_regular_language`.

Abstract We continue the investigations begun in [PRS95] (Intern. J. Computer Math., to appear) on the relationships between several variants of the splicing operation and usual operations with formal languages. The splicing operations are defined with respect to arbitrarily large sets of splicing rules, codified as simple languages. The closure properties of families in Chomsky hierarchy are examined in this context. Several surprising results are obtained about the generative or computing power of the splicing operation. Many important open problems are mentioned.

[LGC$^+$96] Quinghua Liu, Zhen Guo, Anne E. Condon, Robert M. Corn, Max G. Lagally, and Lloyd M. Smith. A surface-based approach to DNA computation. In AMS96 [AMS96].

Abstract: A new model of DNA-based computation is presented. The main difference between this model and that of Adleman is in manipulation of DNA strands that are first immobilized on a surface. This approach greatly reduces losses of DNA molecules during purification steps. A simple, surface-based model of computation is described and it is shown how to implement an exhaustive search algorithm for the SAT problem on this model. Partial experimental progress in solving a 5-variable SAT instance is described, and possible extensions of our model that allow general computations are discussed.

[Lip94] Richard J. Lipton. Speeding up computations via molecular biology, `ftp://ftp.cs.princeton.edu/pub/people/rjl/bio.ps`. Unpublished manuscript Dec. 9, 1994,, December 11, 1994.

Abstract We show how to extend the recent result of Adleman ([Adl94]) to use biological experiments to directly solve any NP problem. We, then, show how to use this method to speedup a large class of important problems.

[Lip95a] Richard J. Lipton. DNA solution of hard computational problems. *Science*, 268:542–545, April 28, 1995.

Abstract: DNA experiments are proposed to solve the famous

"SAT" problem of computer science. This is a special case of a more general method that can solve NP-complete problems. The advantage of these results is the huge parallelism inherent in DNA-base computing. It has the potential to yield vast speedups over conventional electronic-based computers for such search problems.

[Lip95b] Richard J. Lipton. Using DNA to solve NP-complete problems. Technical report, Princeton University, 1995, `http://www.cs.princeton.edu/~dabo/bio-comp/satgen.ps`. , .

[Lip96] Richard J. Lipton. Speeding up computations via molecular biology. In Baum and Lipton [BL96a]. Also known under the working title *DNA Computing*.
Description: This volume presents the proceedings of a conference held at Princeton University on April 4, 1995 as part of the DIMACS Special Year on Mathematical Support for Molecular Biology. The subject of the conference was the new area of DNA based computing. DNA based computing is the study of using DNA strands as individual computers. The concept was initiated by Leonard Adleman's paper in Science in November 1994. Contains [Adl96], [Bau96a], [Bea96], [BDL96], [Lip96], [Rot96], [SS96], [Win95a], [Win95b].
See [Lip94].

[LSW$^+$96] Thomas H. Leete, Matthew D. Schwartz, Robert M. Williams, David H. Wood, Jerome S. Salem, and Harvey Rubin. Massively parallel DNA computation: Expansion of symbolic determinants. In AMS96 [AMS96].
Abstract: A new type of algorithm is introduced for constructing DNA molecules which encode answers to mathematical problems. Examples include problems from the class #P-Complete, which are widely considered to be harder than those in the problem classes previously addressed. In particular, algorithms are presented that generate expansions of symbolic determinants given their patterns of zero entries. This is well-known to be exponentially more difficult than evaluating determinants whose entries are merely numerical. Prior approaches to DNA computation were impractical for large problems because they required processing vast quantities of DNA with steps associated with large error propagation. Our new approach to the production of the solution and reading the answer is based on

reliable and automatable PCR steps and can solve large problems by processing up to 10^{15} or more distinct strands of DNA in parallel. The DNA algorithms described here should be applicable to a wide variety of problems that are intractable using conventional computers.

[MD96] Dónall 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, http://www.iicm.edu/jucs_2_2/on_the_scalability_of/ps/paper.ps;internal%&sk=ROBOT.

Abstract: A molecular computational procedure in which manipulation of DNA strands may be harnessed to solve a classical problem in NP — the directed Hamiltonian path problem — was recently proposed [Adl94, Gif94]. The procedure is in effect a massively parallel chemical analog computer and has a computational capacity corresponding to approximately $\approx 10^5$ CPU years on a typical 10 MFLOP workstation. In this paper limitations on the potential scalability of molecular computation are considered. A simple analysis of the time complexity function shows that the potential of molecular systems to *increase* the size of generally solvable problems in NP is *fundamentally* limited to $\approx 10^2$. Over the chemically measurable picomolar to molar concentration range the greatest practical increase in problem size is limited to $\approx 10^1$.

Reiterates the argument of [Har95b, Har95a] that Molecular Computation brings nothing new to the theory of computational complexity: it cannot break the exponential barrier.

[Mir96] Kalim U. Mir. A restricted genetic alphabet for DNA computing. In AMS96 [AMS96].

Introduction: Since Adleman demonstrated his original groundbreaking scheme [Adl94], a simpler approach suggested by Lipton [Lip95a] has widened the range of problems that can be addressed by DNA computing. A single molecular operation, DNA annealing, is required for Lipton's scheme. This also forms the basis of Baum's proposal for a content-addressable DNA memory [Bau95]. In both cases an extractor or cue oligonucleotide, most likely to be in the solid-phase, attached to beads, would anneal to a longer single-stranded target present in the graph or memory. So far, most work on DNA computing has rightly concentrated on what is theoretically possible. Here however, I will discuss some practical issues and offer a means to overcome some practical obstacles.

[Oli96] John S. Oliver. Computation with DNA-matrix multiplication. In AMS96 [AMS96], `http://www.chem.brown.edu/brochure/people/jso/DNA.html`.
Abstract: If chemical reactions are to be used as the basis for computers, efficient instruction sets will need to be developed. A chemically based computation can not at this time be expected to compete with an electronic computer. However, the potential usefulness of a chemical computer provides a compelling reason to investigate and design procedures for the solution of varied problems. DNA based methods which may be used to calculate the product of Boolean matrices or matrices containing positive, real numbers are represented. This provides a method to perform a quantitative calculation with DNA.

[Pău95] Gheorghe Păun. Computationally universal distributed systems based on the splicing operation. Submitted, 1995.

[Pău96a] Gheorghe Păun. Five (plus two) universal DNA computing models based on the splicing operation. In AMS96 [AMS96].
Abstract: We briefly present five types of mechanisms (and we mention two other related devices) based on the *splicing operation* (a model of the recombinant behavior of DNA sequences under the influence of restriction enzymes and ligases). All these models characterize the recursively enumerable languages, hence all are equal in power to the Turing machines. On the basis of the constructions in the proofs of this assertion, one can obtain *universal* (hence programmable) computing devices.

[Pău96b] Gheorghe Păun. Regular extended H systems are computationally universal. *Journal of Automata, Languages, Combinatorics*, 1(1):27–36, 1996.

[PRS95] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. Restricted use of the splicing operation. Technical Report

TR95-16, Department of Computer Science, Leiden University, P.O. Box 9512, 2300 RA Leiden, The Netherlands, June 1995.

Abstract Splicing is a new powerful tool, stemming originally from molecular genetics but investigated extensively also in language theory. In this paper we investigate variants of splicing inspired partly by regulating mechanism customarily studied in language theory, partly by imposing restrictions on the pairs to be spliced or on the result of splicing. The Chomsky hierarchy constitutes a very suitable test bed for the resulting families, because it is classical and well understood. In contrast to the usual, nonrestricted splicing, we find several cases when the families of regular or of context-free languages are not closed under the new types of splicing. On the other hand, our results give new characterizations for families in the Chomsky hierarchy and for closure properties in general.

[PRS96] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. Computing by splicing. *Theoretical Computer Science*, 168(2):321–336, 1996.

[Rei95] John H. Reif. Parallel molecular computation: Models and simulations. In *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA95), Santa Barbara, June 1995*, pages 213–223. Association for Computing Machinery, June 1995, `http://www.cs.duke.edu/~reif/paper/mole.ps`, `http://www.cs.duke.edu/~reif/paper/mole.fig.ps`.

[Res] BioTech Resources. Biotech's on-line dictionary of biotechnology, `http://biotech.chem.indiana.edu/pages/dictionary.html`.

[Rot96] Paul Wilhelm Karl Rothemund. A DNA and restriction enzyme implementation of Turing machines. In Baum and Lipton [BL96a], `http://www.ugcs.caltech.edu/~pwkr/oett/dimacs/dimacs.ps`, `http://www.ugcs.caltech.edu/~pwkr/oett.html`.

Abstract Bacteria employ restriction enzymes to cut or *restrict* DNA at or near specific words in a unique way. Many restriction enzymes cut the two strands of double-stranded DNA at different positions leaving overhangs of single-stranded DNA. Two pieces of DNA may be rejoined or *ligated* if their terminal overhangs are complementary. Using these operations fragments of DNA, or oligonucleotides may be inserted and deleted from a circular piece of plasmid DNA. We propose an encoding for the transition table of a Turing machine in DNA oligonucleotides and a corresponding series of restrictions and ligations of those oligonucleotides that, when performed on circular DNA encoding an instantaneous description of a Turing machine, simulate the operation of the Turing machine encoded in those oligonucleotides. DNA based Turing machines have been proposed by Charles Bennet but they invoke imaginary enzymes to perform the stat-symbol transitions. Our approach differs in that every operation can be performed using commercially available restriction enzymes and ligases.

A very detailed scheme for simulation Turing machines in DNA. Provides references to papers prior to [Adl94] containing some of the ideas of Molecular Computation.

[RS96] Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages.* Springer Verlag, Berlin, Heidelberg, New York., October 1996, ISBN Vol 1: 3-540-60420-0, Vol 2: 3-540-60648-3, Vol 3: 3-540-60649-1.

Description

Volume 1: Word, Language, Grammar

This first volume of the Handbook of Formal Languages gives a comprehensive authoritative exposition on the core of language theory. Grammars, codes, power series, L systems, and combinatorics on words are all discussed in a thorough, yet self-contained manner. This is perhaps the most informative single volume in the history of theoretical computer science.

Volume 2: Linear Modeling: Background and Application

This second volume of the Handbook of Formal Languages contains the most fundamental applications of language theory. Various aspects of linguistics and parsing, both natural and programming languages, symbolic manipulation, and pattern matching are discussed. A special feature is the recently very active field of DNA computing.

Volume 3: Beyond Words

This third volume of the Handbook of Formal Languages discusses

language theory beyond linear or string models: trees, graphs, grids, pictures, computer graphics. Many chapters offer an authoritative self-contained exposition of an entire area. Special emphasis is on interconnections with logic.
Volume 2 contains [HPP96].

[RW95] Diana Rooß and Klaus W. Wagner. On the power of DNA-computers. Technical report, University of Würzburg, 1995, ftp://haegar.informatik.uni-wuerzburg.de/pub/TRs/ro-wa95.ps.gz.
Abstract In [Adl94] Adleman used biological manipulations with DNA strings to solve some instances of the Directed Hamiltonian Path Problem. Lipton [Lip94] showed how to extend this idea to solve any NP problem. We prove that exactly the problems in $P^{NP} = \Delta_2^p$ can be solved in polynomial time using Lipton's model. Various modifications of Lipton's model are investigated, and it is proved that their computational power in polynomial time can be characterized by one of the complexity classes P, Δ_2^p, Δ_3^p or even PSPACE. Restricting Liptons model to DNA strings of logarithmic length one can compute exactly the promblems in L.

[RWB⁺96] Sam Roweis, Erik Winfree, Richard Burgoyne, Nickolas V. Chelyapov, Myron F. Goodman, Paul W. K. Rothemund, and Leonard M. Adleman. A sticker based architecture for DNA computation. In AMS96 [AMS96], ftp://hope.caltech.edu/pub/roweis/DIMACS/stickers.ps.
Abstract: We introduce a new model of molecular computation that we call the *sticker* model. Like many previous proposals it makes use of DNA strands as the physical substrate in which information is represented and of separation by hybridization as a central mechanism. However, unlike previous models, the stickers model has a random access memory that requires no strand extension, uses no enzymes, and (at least in theory) its materials are reusable. The paper describes computation under the stickers model and discusses possible means for physically implementing each operation. We go on to propose a specific machine architecture for implementing the stickers model as a microprocessor-controlled parallel robotic workstation. Finally, we discuss several methods for achieving acceptable overall

error rates for a computation using basic operations that are error prone. In the course of this development a number of previous general concerns about molecular computation [SS95, Har95b], [Letters to Science] are addressed. First, it is clear that general-purpose algorithms can be implemented by DNA-based computers, potentially solving a wide class of search problems. Second, we find that there are challenging problems, for which only modest volumes of DNA should suffice. Third, we demonstrate that the formation and breaking of covalent bonds is not intrinsic to DNA-base computation. This means that costly and short-lived materials such as enzymes are not necessary, nor are energetically costly processes such as PCR. Fourth, we show that a single essential biotechnology, sequence-specific separation, suffices for constructing a general-purpose molecular computer. Fifth, we illustrate that separation errors can theoretically be reduced to tolerable levels by invoking a trade-off between time, space, and error rates at the level of algorithm design; we also outline several specific ways in which this can be done and present numerical calculations of their performance. Despite these encouraging theoretical advances, we emphasize that substantial engineering challenges remain at almost all stages and that the ultimate success or failure of DNA computing will certainly depend on whether these challenges can be met in laboratory investigations.

[Sap58] E. Sapir. *The Status of Linguistics as a Science.* University of California Press, Berkeley, CA, USA, 1929/1958.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings, 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE Computer Society Press, 1994, `ftp://netlib.att.com/netlib/att/math/shor/quantum.algorithms.ps.Z`.

[Smi95] Warren D. Smith. An opinionated, but reasonably short, summary of the Mini DIMACS workshop on DNA based computers, (held at Princeton University on April 4 1995), April 5 1995, `http://www.neci.nj.nec.com/homepages/smith/workshop.summary.ps`.

[SS95] Warren D. Smith and Allan Schweitzer. DNA computers in vitro and vivo. Technical report, NEC Research Institute, March 20, 1995. Manuscript of 3/20/95, presented at DIMACS Workshop on DNA Based Computing, Princeton, 4/4/95.

Abstract We show how DNA molecules and standard lab techniques may be used to create a nondeterministic Turing machine. This is the first scheme that shows how to make a universal computer with DNA. We claim that both our scheme and previous ones will work, but they probably cannot be scales up to be of practical computational importance. In vivo, many limitations on our and previous computers are much less severe or do not apply. Hence, lifeforms ought, at least in principle, to be capable of large Turing universal computations. The second part of our paper is a loose collection of biological phenomena that look computation and mathematical models of computation that look biological. We observe that cells face some daunting computational problems, e.g., gene regulation, assembly of complex structures and antibody synthesis. We then make simplified mathematical modelsof certain biochemical processes and investigate the computational power of these models. The view of “biology as a computer programming problem” that we espouse, can be useful for biologists. Thus our particalar Turing machine construction bears a remarkable resemblance to (and probably explains) recently discovered “RNA editing” processes. In fact it may be that the RNA editing machine in T. Brucei is clonable, extractible and runnable in vitro, in which case one would have a better performing Turing machine than with our construction. The fact that RNA editing is a Turing machine may in turn have a lot to do with the origins of life. We also have a possible explanation for “junk DNA”.

[SS96] Warren D. Smith and Allan Schweitzer. DNA computers in vitro and vivo. In Baum and Lipton [BL96a].
See [SS95].

[Ste95] Willem P.C. Stemmer. The evolution of molecular computation. *Science*, 270:1510–1510, December 1, 1995.
Molecular computation in the style of [Adl94] and [Lip95a] requires too much DNA even for rather small problem instances. Nature has sought through such a large search space using a much smaller pool of

sequences, by evolution: repeated cycles of selection from small pools. The author suggests to use similar methods in attacking problems using molecular computation: approximate solutions by treating a problem with a dynamic programming approach.

[SWL+96] Nadrian C. Seeman, Hui Wang, Bing Liu, Jing Qi, Xiaojun Li, Xiaoping Yang, Furong Liu, Weiqiong Sun, Zhiyong Shen, Ruojie Sha, Chengde Mao, Yinli Wang, Siwei Zhang, Tsu-Ju Fu, Shouming Du, John E. Mueller, Yuwen Zhang, and Junghuei Chen. The perils of polynucleotides: The experimental gap between the design and assembly of unusual DNA structures. In AMS96 [AMS96].

Abstract: DNA computing relies on the successful implementation of physical chemistry techniques involving oligonucleotides of prescribed sequence. Our laboratory has been involved in genetic recombination and nanofabrication. We have constructed a large number of unusual DNA molecules, including branched DNA molecules, DNA polyhedra, DNA knots, DNA double crossover molecules, and DNA antijunctions and mesojunctions. Our experience with these systems has uncovered a large number of experimental pitfalls that may confront individuals working with DNA computing. We present our experience in this area with the hope that we can help investigators to anticipate the experimental problems that may affect their DNA computing schemes.

- DNA computing will have to use physical chemistry in order to get substantial — detectable — yields of desired results. The molecular biological techniques (e.g. PCR) are not sufficient since they only work well when sequence properties of the desired solution are known; this is generally not the case.
- The main problem in building unusual DNA structures is one of control. In general, multiple outcomes are (nearly) equivalent from the standpoint of free energy. The undesired alternatives must be made sufficiently unfavourable in relation to the target. Since — under certain reaction circumstances — Watson-Crick bonds are highly favoured, one can try to choose the base sequence in such a way that Watson-Crick pairing favours the intended design.
- Reaction circumstances are very important.
- The 3D structure of molecules is important (e.g. the twist in the double helix), as is the flexibility of the structure.
- Some ligases used to ligate sticky ends are "hungry" and will settle for an end that is close to its optimal one.

[Who40] B.L. Whorf. Science and linguistics. *Technology Review*, 42(6), 1940.

[Win95a] Erik Winfree. Complexity of restricted and unrestricted models of molecular computation. In Baum and Lipton [BL96a], `http://dope.caltech.edu/winfree/Papers/models.ps.gz`.
Description: Here I show some limits on what can be computed using some proposed operations on DNA. These limits have since been overcome by the inclusion of additional operations.
Abstract In [Lip94] and [Adl94] a formal model for molecular computing was proposed, which makes focused use of affinity purification. The use of PCR was suggested to expand the range of feasible computations, resulting in a second model. In this note, we give a precise characterization of these two models in terms of recognized computational complexity classes, namely branching programs (BP) and nondeterministic branching programs (NBP) respectively. This allows us to give upper and lower bounds on the complexity of desired computations. Examples are given of computable and uncomputable problems, given limited time.

[Win95b] Erik Winfree. On the computational power of DNA annealing and ligation. In Baum and Lipton [BL96a], `http://dope.caltech.edu/winfree/Papers/ligation.ps.gz`.

Description: Here I show how one might create a "one-pot" mixture of DNA which can perform universal computation. A.k.a. "weaving the tapestry of life". [Note, there are strand polarity errors in several figures. EW, 5/96]

Abstract In [Win95a] it was shown that the DNA primitives of *Separate*, *Merge* and *Amplify* were not sufficiently powerful to invert functions defined by circuits in linear time. Dan Boneh et al [BDS96] show that the addition of a ligation primitive, *Append*, provides the missing power. The question becomes, "How powerful is ligation? Are *Separate*, *Merge*, and *Amplify* necessary at all?" This paper proposes to informally explore the power of annealing and ligation for DNA computation. We conclude, in fact, that annealing and ligation alone are theoretically capable of universal computation.

[WW96] Robert M. Williams and David H. Wood. Exascale computer algebra problems interconnect with molecular reactions and complexity theory. In AMS96 [AMS96].

Abstract: In discussing exascale (exa = 10^{18}) computer algebra problems we interconnect three themes. First, DNA is an attractive medium for computation because of its density and parallelism. Second, computer algebra is similar to DNA laboratory reactions. Both rearrange identical subunits. Third, determinant and/or permanent expansions exemplify many levels of complexity. These three issues are combined in a planned experiment using a DNA algorithm to evaluate or approximate the permanent of a matrix of zeros and ones, a well-known problem in the class #P-Complete. Such problems are harder than those previously addressed by DNA techniques in the pioneering articles of Adleman and Lipton. This points the way to DNA methods for expanding a symbolic determinant given its zero pattern, which is of still higher complexity. We begin to approach interesting problem sizes because we reduce scale-up difficulties by alternating intermediate steps of building and filtering. The example algorithm suggests directions toward the general problem of expanding symbolic determinants and permanents given their zero entries.

[WYS96] Erik Winfree, Xiaoping Yang, and Nadrian C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In AMS96 [AMS96], `ftp://hope.caltech.edu/pub/winfree/DIMACS/self-assem.ps`. Draft.

Abstract: In this paper we examine the computational capabilities inherit in the hybridization of DNA molecules. First we consider theoretical models, and show that the self-assembly of oligonucleotides into linear duplex DNA can only generate sets of sequences equivalent to regular languages. If branched DNA is used for self-assembly of dendrimer structures, only sets equivalent to context-free languages can be achieved. In contrast, we show that the self-assembly of double crossover molecules into two dimensional sheets or three dimensional solids is theoretically capable of universal computation. The proof relies on a very direct simulation of a universal class of cellular automata. In the second part of this paper, we present results from preliminary experiments which investigate the critical computational step in a two-dimensional self-assembly process.

[YKF95] Takashi Yokomori, Satoshi Kobayashi, and Claudio Ferretti. On the power of circular splicing systems and DNA computability. Technical Report Report CSIM 95-01, University of Electro-Communications, Department of Computer Science and Information Mathematics, Chofu, Tokyo 182, Japan, July 1995.

Abstract A new type of generative mechanisms was recently introduced under the name of extended H systems, and it has been shown that extended H systems with finite sets of axioms and finite sets of rules exactly characterize the recursively enumerable languages, thus having the full power of Turing machines. Also, it was shown that there is a universal extended H system analogous to a universal Turing machine. In this paper, we propose a new type of splicing models called circular H systems, and show that they have the same computational power as Turing machines. Proposed new models are based on circular splicings which come from biological motivations of interactions between linear and circular DNA sequences, and hence, the models seem to have some advantages over other existing models dealing with only linear strings. We also show that there effectively exists a universal circular H system which can simulate any circular H system with the same terminal alphabet, which naturally leads us to a feasible design for a DNA computer based on circular splicing. Surprisingly, all these results are obtained without considering multiplicity constraints, which is in marked contrast to the previous results for linear H systems.

Note. The bibliography database on Molecular Computation and Splicing Systems [Das] collected during the research for this thesis is available online. The author is not aware of any bibliography of comparable or larger size on these subjects and expresses his hope that contributions of its users will keep it accurate, complete and up to date.

Index