Giuditta Franco

Biomolecular Computing – Combinatorial Algorithms and Laboratory Experiments

Ph.D. Thesis

21st April 2006

Università degli Studi di Verona
Dipartimento di Informatica

Advisor:
prof. Vincenzo Manca

Series N°: **TD-04-06**

Università di Verona
Dipartimento di Informatica
Strada le Grazie 15, 37134 Verona
Italy

A mia madre

Contents

Preface

I first heard of molecular computing at the Department of Biology in Pisa, where I was hanging out, as an undergraduate student of mathematics, with the firm intent to learn a certain programming language. I ended up in a lecture of Professor Vincenzo Manca (who would later become the supervisor of this thesis), and it was amazing. Researchers were trying to compute in a novel manner, by processing information stored in biomolecules according to innovative paradigms, and to "decipher" computations taking place in nature, by observing biological processes from a mathematical perspective. A speculative investigation which genuinely combines an interest in living beings with the logic behind the abstract perfection of mathematics, seemed to me extremely attractive. Moreover, the chance to analyze an alternative concept of computation inspired by nature stimulated my curiosity and my imagination. I have not learned that programming language as yet, but after that lecture I continued to attend the whole class, thus starting my enthusiastic studies in molecular computing and (theoretical) computer science.

This thesis collects most of my research in molecular computing, according to the chronological order in which it was conceived, while some additional contributions are briefly mentioned along the way. The original content of the thesis has been almost entirely published or submitted for publication, and the papers are respectively listed at the end of the chapters that are based on them. The reader may also find some more recent ideas in Chapter 8, where work in progress is outlined. Most of the results presented arose from collaboration, mainly with my advisor and his group, but also with people coming from other departments (computer science, mathematics) and from medical schools. For this reason, the academic *we* is often preferred to *I*.

After an introduction in Chapter 1, which offers a condensed overview of the literature and current trends in molecular computing, with more details in DNA computing, together with a presentation of the problems faced in this dissertation, we proceed through three research phases.

Firstly, we ponder why the DNA molecule is assembled as a pair of filaments of opposite directions exhibiting Watson-Crick complementarity between the corresponding bases. What differences would we find, if it were differently structured? Why antiparallelism? Why just four bases? Since we did not find any biological account for these facts, we looked for an explanation in informational or computa-

tional terms. An algorithmic analysis of DNA molecular features addresses these questions in Chapter 2, and the double stranded DNA structure turns out to be a requirement necessary for the efficiency of the DNA autoduplicating process. A DNA-like duplication algorithm is then extended to universal computations on double strings, while a general presentation of membrane systems concludes the chapter. In particular, in membrane computing, a different computational perspective is suggested, that shifts the focus of attention from the evolution of individual objects to that of a population of objects. The same approach is useful for laboratory DNA computations, where one is generally unable to know exactly what happens to an arbitrary single strand, but it is possible to manipulate a *pool* of strands, in such a way that after a computation the pool satisfies a certain property, as was the case in the experiments reported in Chapter 6.

Secondly, we perform computations with DNA strings *in vitro* (in contrast to *in vivo*) both in theoretical (dry) and experimental (wet) contexts. A variant of the Polymerase Chain Reaction, which is perhaps the most widely applied technique in molecular biology, is introduced as an implementation basis for generation, extraction and mutagenesis procedures. Novel applications, which improve methods to be found in the literature, are presented by means of bioalgorithms that are based on polymerase action. Theoretical investigations of a specifically guided generation of DNA libraries are described, as well as laboratory experiments that validate those algorithms. Namely, Chapter 4 showcases a simple example of how mathematical results may be motivated from, and prove significant for, an experiment in DNA computing.

Finally, we broaden our interest from linear self-assembly of double strings to a three-dimensional self-assembly process. Recently this has been the subject of extensive studies, since the construction of complex structures at the nano scale is the key challenge at the core of the emerging discipline of Nanoscience. The possible outcomes of the self-assembly process must comply with constraints which arise from the physical and chemical properties of DNA, and are described in Chapter 7 by a set of forbidding and enforcing rules. Forbidding-enforcing systems are an alternative device to generate formal languages that was inspired by chemical processes. This device was put to work to simulate DNA-based computations, and afterwards it was introduced to the context of membrane computing. A variant of forbidding-enforcing systems for graphs is proposed here, modelling the DNA self-assembly process and suggesting new research directions in graph theory.

The three phases outlined above are put forth as more or less selfcontained stages of a research journey, the main results being found in the middle one. I hope this arrangement makes for a smoother access by those readers who are interested in just some parts of this dissertation.

Verona, March 31, 2006

1

Introduction

Computer Science is no more about computers
than astronomy is about telescopes.

Edsger Wybe Dijkstra

Computer science is a pretty reductive name to connote the discipline of information processing, unless we take our mind away from the machines populating our everyday life and we think of a 'computer' just as an apparatus performing some kind of computation. More precisely, the attention should be focused on the nature of computation more than on the device performing computation. On the other hand, concepts as information, processes, computations, algorithms are so wide and pervasive that some context seems necessary to put any problem relating to the nature of computation in the right focus.

Information is one of the fundamental building blocks of the universe, and a computation is a process that transforms information. This looks like a nice presentation, but it says all and nothing. I prefer to introduce the idea of information as an analogue of the well known notion of energy[1]. Much like energy is expressed by the motion of matter and by its changing of form, analogously information is manifested by the transformation of data, that is, by the change from a representation in a certain kind of data into another one, within another kind of data. To give a more evident analogy, as energy also is the potential to perform work, so information also is the potential to rule the performance of work. In fact, the performing of work releases energy according to some constraints, and information establishes such constraints to select some work processes out of all the possible ones.

If we think of information as being numerical, statistical, qualitative, quantitative, symbolic, biological, genetic, linguistic, artistic, then we realize how the idea of information is deeply related to concepts of action, communication, data, form, representation[1]. Although it is an abstract entity, we cannot study its 'being

[1] This idea is proposed and widely discussed along with interesting examples in Chapter 6 of the book: V. Manca, *Metodi Informazionali*, Bollati Boringhieri, 2003.

processed' without taking the physical supports on which it is represented into account. That is to say, we may keep track of the transformation, transmission, conservation, deterioration of information expressed by data represented, for example, on paper, electronic or magnetic supports, or molecules. Hence, we need to consider the structure as well as the location in space and the temporal life of such supports. Note that this approach assumes the discrete and quantitative nature of data, underlying digital information (in contrast to analogical information, given by continuous measures that are then approximated by discrete quantities).

The processing of information requires a strategy of systematical data transformation by means of basic operations called computations. The definition of such basic operations depends on the type of data to which they apply, as well as on their organization within physical supports. An algorithm is such a strategy (more intuitively, in [9] it is characterized by P. Arrighi as "a recipe to solve a mathematical problem"), and a computational process is the procedure composed of the computations running onto a specific system. We shall use the words 'algorithm' and 'computational procedure' indifferently, because it will be clear from the context which system do they run on and what kinds of computations do (bio-) algorithms refer to.

The concepts of algorithm and computation emerged even earlier than the idea of mathematical proof, which was introduced by the ancient Greeks along with the axiomatic deductive method of Euclidean geometry. Those concepts may already be found in the first documents of ancient civilizations, connected to problems of measurements of land plots or altars[2]. The Ahmes book (XVII century AC) is a rich example, where the algorithmic solutions of eighty problems were elaborated by Egyptian mathematicians, and comparable works can be retrieved from ancient Arabic, Indian and Chinese civilizations[3]. The idea of computational model, on the contrary, was only introduced in the past century, along with the Mathematical Logic of the thirties [227], where the concept of computable function arose from the computational equivalence of such diverse formalisms as the λ-calculus by Alonzo Church and the computing machine by Alan M. Turing. The von Neumann architecture was influenced by Turing (they are computationally equivalent by universality theorem); it is based on the principle of one complex processor that sequentially performs a single task at any given moment, and has dominated computing technology for the past sixty years. Recently, however, researchers have begun exploring alternative computational systems inspired by biology and based on entirely different principles.

Nowadays, are new theories of computation worth the effort of their creation? There are several reasons to believe the effort has been well spent [130]. First of all, it is important to determine the conditions under which the present theory of computation holds. Our present notion of effective computability may be unnecessarily limited, namely if other systems subsume the class of general recursive functions [221] and an expanded definition is as useful as the current one for abstract problems. A more physically and biologically grounded theory would likely

[2] A historical *excursus* regarding algorithmic theory is given in the book: V. Manca, *Note di teoria degli algoritmi*, Cooperativa Libraria Universitaria Friulana, 1992.

[3] B. L. Van Der Waerden. *Geometry and Algebra in Ancient Civilizations*, Springer-Verlag, Berlin-Heidelberg, 1983.

stimulate substantial changes in our understanding of computing and in the design of computers. A theory that could enable the design of better algorithms and of molecular computers, would distinctly improve the current situation.

First electronic computers date to the end of the forties, when the molecular biology discipline was just born. Since then, there have been speculations about exploiting molecules as computational devices [166,233], and first molecular computational models have been proposed [133, 65, 45, 15, 16, 46]. As an example, C. Bennett in [15] discusses how RNA could be used as a physical medium to implement (reversible) computation, by linking the functioning of RNA polymerase to the one of a Turing machine, whereas in [16] he gives a description of a hypothetical computer, using a macromolecule similar to RNA to store information, and imaginary enzymes to catalyze reactions acting upon the macromolecule. As a matter of fact, computer science and molecular biology have as clear as surprising analogies in the data structures on which their fundamental processes are based. Only recently, though, biological computation has become a reality, thanks to remarkable developments both in molecular biology and in genetic engineering, now proven able to sequence the human genome, to measure enormous quantities of data accurately, and to synthesize and manipulate considerable amounts of specific molecules.

The breakthrough was the experiment which Leonard M. Adleman carried out in November 1994, thereby launching a novel, in vitro approach to solve a Hamiltonian Path Problem (HPP) instance, over a directed graph with seven vertices, by means of DNA molecules and of standard biomolecular operations [1]. Given a graph and a pair of designated 'initial' and 'final' vertices, the goal of HPP is to determine whether a path exists, starting from the initial vertex and ending up in the final one, such that it goes through each vertex of the graph exactly once. While in conventional computers information is stored as binary numbers in silicon-based memories, in this approach he encoded the information by random DNA sequences of twenty bases. The computation was performed in a fashion of biomolecular reactions, involving procedures such as hybridization, denaturation, ligation, polymerase chain reaction, and a biotin-avidin magnetic beads system. The output of computation, in the form of DNA molecules, was read and printed by means of a standard electrophoresis fluorescence process.

This seminal experiment had a tremendous impact onto the information science community, in that it stimulated a huge avalanche of new and exciting research, dealing with both experimental and theoretical issues of computing by molecules. A new philosophy promises to deliver new means for doing computation more efficiently, in terms of speed, cost, power dissipation, information storage, and solution quality. Simultaneously, it offers the potential of addressing many more problems, also coming from the biological world, than it was previously possible, and the results obtained so far hold prospects for a bright future.

1.1 Molecular computing

In the past fifteen years a fast growing research field called molecular computing emerged in the area of natural computing. The general aim of research in this

area, is to investigate computations underlying natural processes, or taking place in nature, as well as to conceive new computational paradigms and more efficient algorithms inspired by nature. Significant advances in this area have been made through neural networks, evolutionary algorithms, and quantum computing, that are examples of human-designed computing inspired by nature. In particular, information massively encoded in molecules nowadays can be extensively processed by technologies from molecular biology, and a new type of computation has been actually initiated ("Most importantly, this research has led already to a deeper and broader understanding of the nature of computation", G. Rozenberg, [201]).

Biological strings can be obviously seen as strings over an alphabet (having four symbols in the case of DNA and RNA polymers), and some operations can be quite naturally imported from formal language theory [202]—which is a mathematical theory of strings, essentially. It is not so surprising, though, that when moving to the laboratory reality of a test tube, where strings really float and interact, we lose most of the mathematical assumptions behind formal language theory. For example, the operations of concatenating, reading, and writing strings on molecules require specific bioalgorithms and ad hoc biotechnological protocols, as well as other simple operations, like computing their length or 'taking a string from a set to which it belongs'. Universal computations of the Turing machine can be simulated at a molecular level (for a recent reference see [174]), in a different fashion though, where selective cutting, pasting (concatenating), pairing and elongating strings are natural operations, while the moving of a reader along a string is more specific and limited to the action of some enzymes—quite unlike what happens on a tape. Moreover, nature does not 'compute' just by sequence rewriting but also by crossing-over, annealing, insertion-deletion and so on. Nonetheless, while reading and writing are not so obvious, parallelism of computations and compactness of information storage are given for free by nature, and the limits to miniaturization of silicon technology do not exist anymore. Interestingly enough, novel algorithms are thus designed in order to make molecular computations robust, that are based on different, appropriate kinds of operations.

As it emerged from discussions with Vincenzo Manca, an important new feature which is common to molecular and quantum computations is that they can be observed and captured only at certain moments. In both cases indeed, one cannot control step by step what is the exact effect of the operations, and one alters the system when observing the result. The sound way to account for what happens between the moment of preparing an initial system and the moment of measuring it is to think of the complex components of the state as amplitudes, or proportions (rather than probabilities). As an example, a quantum coin can be in a state of both head and tail, in some proportions, simultaneously, until one measures it. This feature is just a means of exploring several possibilities simultaneously [9]. The case of DNA operations is interesting as well, they are performed on *pools*, identified with multisets (sets with a multiplicity associated to their elements) of strings over a given alphabet, but we do not really know which are the exact multisets before and/or after the application of the operations. In fact, to this purpose we would need to estimate how many copies of each different molecule are present at any given moment, and how many of them are modified by the operation as expected, but this claim is not realistic at all, because we are only

able to observe macro-states of the pool. Realistically speaking, the output pool is a mixed population of 'solutions' and 'other outcomes', and no theory of molecular computation may presume a single unambiguous, exact result of computation.

Despite this kind of approximation, that we could say microscopic nondetermination of the computation, bioalgorithms driving system behaviours can be written such that the final pool of molecules contains the desired solution [150]. This fact uncovers a different perspective of computing: one does not care what happens exactly to each sequence in each test tube, but one knows that an operation applied to certain data yields a pool of data satisfying a certain property. The individual nondeterminism is left unaffected, while population properties are controlled by macro-steps. This approach seems to be a best way to really manipulate computations in nature, rather than trying to fit the (mysterious) natural processes into pre-conceived, exact computational models.

Nevertheless, no new model may escape the comparison with traditional computational models; and a significant research stream in molecular computing aimed at molecular implementation of classical models of computation. For example, molecular implementation of Turing machines was considered in the articles [15] and [198], where detailed molecular encodings of the transition table of a Turing machine are given, and further Turing machines with few instructions can be found in [195]. Molecular implementations of linear context-free grammars [59], Post systems [63], cellular automata [242], and push-down automata [39] were discussed as well. Maybe more interestingly, new computational models were introduced and extensively investigated, such as *aqueous computing*, *forbidding-enforcing computing*, and *membrane computing*.

1.1.1 Unconventional computational models

Tom Head, in [101], introduced a framework of aqueous computing as a kind of molecular computing free from code design, while the first aqueous computation to be completed was done in Leiden by operating on plasmids [104]. This work solved the problem of computing the cardinal number of a maximal independent subset of the vertex set of a graph, by dealing with the same instance faced in [175]. In general, aqueous computing requires both water-soluble molecules on which a set of identifiable locations can be specified, and a technology for making local alterations and for detecting the condition of molecules. A vast number of such molecules are dissolved in the 'water' (i.e., an appropriate fluid where the chosen molecules are soluble), and the initial state of each of the specified locations takes a bit value. At this point one can 'write on the molecules' by altering such bit values: the first phase of computation always consists of a sequence of writing steps. After suitable separations and unifications of pools of molecules, the result of the computation is 'read' from the state of the molecules. More recent examples of aqueous computing can be found in [102, 225].

Andrzej Ehrenfeucht, Hendrik Jan Hoogeboom, Grzegorz Rozenberg and Nikè van Vugt, in [55], introduced the forbidding-enforcing model that is inspired by chemical reactions, where the presence of a certain group of molecules implies the presence of other specific molecules eventually, while 'conflicting' reactants may not be present simultaneously in the system. This model was novel both

from the viewpoint of molecular systems and from that of computation theory, in that it suggested a new way to compute a whole family of outcomes, all of which obey the forbidding and enforcing constraints of the system [58]. The result of an evolving computation of the system can be, for example, a possibly infinite family of languages, or the result of interactions between DNA molecules, but the computational model is not restricted to operations on strings. Indeed, it was applied to membrane computations [37], and also to describe some processes of DNA self-assembly as computations on graphs [74]. More details about this model are reported in Section 7.1.

Gheorghe Păun, in [178], introduced the first class of discrete models directly inspired by living cell structure and functioning. They were named P-systems, after their "inventor", or membrane systems, because of their hierarchically arranged structure of membranes, all embedded inside one, outermost "skin membrane". This compartmentalization in regions delimited by (and one by one associated to) membranes is a crucial point of the model, to compute in a parallel and distributed way. Each membrane region is associated to a multiset of objects which evolve according to the set of rules relating to that region. The evolution of (each region of) the system is performed in a maximally parallel way, that is by nondeterministically assigning objects to the rules until no further assignment is possible. All compartments of the system evolve synchronously (a common clock is assumed for all membranes), therefore there are two layers of parallelism: between the objects inside the membranes and between the membranes inside the system. A computation starts from an initial configuration of the system, that is defined by a cell-structure with objects and a set of rules for each compartment, and goes on by applying rules as described above, until it reaches a configuration where no more rules apply to the existing objects. The move from a configuration of the system to a next one is called transition, and a computation is a sequence of transitions. If such a sequence is finite, then it is a halting computation and, because of the nondeterminism of rule application, we get a set of results (a language or a set of numerical vectors), in a designated (output) membrane.

Membrane computing is thus a framework for distributed parallel processing of multisets inspired by cell biochemistry. In the original definition, three types of membrane systems were considered: transition, rewriting and splicing, depending on the kinds of objects and on the way they are processed. However, the generality of this approach is evident; objects can be symbols, strings, trees, and the rules can be associated to the regions, or also to the membranes, and can be of different type, namely, rules of object transformation or communication (when objects move through membranes to adjacent compartments), of membrane creation or dissolution. Models inspired by the way cells are organized to form tissue were considered as well, as a variant of P-systems. More details about membrane systems are recalled in Section 2.5.

From a theoretical point of view, membrane computing is an extension of molecular computing, where objects represent molecules and rules represent molecular computations. Since the computation proceeds in a distributed, parallel manner, membrane systems model well both in vivo and in vitro molecular computations ("Decentralization, random choices, nondeterminism, loose control, asynchronization, promoting/inhibiting, control paths are key-words of the 'natural computing',

of the manner the cell preserves its integrity, its life, but they are still long term goals for computer science *in silico*, a challenge for computability in general." Gh. Păun, in [181]).

Usually, biomolecular computations are implemented in vitro, hence outside of living cells, by manipulating DNA and related molecules moving around test tubes or compartments of a membrane system. However, some attempts in vivo have been made; for example, boolean circuits were simulated in living cells by conducting in vivo evolution of RNA switches by a self-cleaving ribozyme [232], and, more recently, finite-state automata based on a framework which employs the protein-synthesis mechanism of Escherichia Coli have been implemented in [173]. Regarding molecular computations in vitro, although there are interesting examples of RNA-based computing [137, 61], the DNA support has been more widely exploited and investigated.

1.1.2 Why choosing DNA to compute?

First of all, one should wonder why molecular computing is focused on biomolecules rather than other kinds of molecules existing in nature[4]. Certainly, strings are useful to encode discrete data and to compute, and natural polymers are especially apt to this purpose. Besides, by having strings as data structures, one can apply (or at least relate molecular computing to) all the results from formal language theory, which underlies the theory of computation, and from information theory. In Section 1.1 the main features of biological strings are described. The reason why our attention is focused on strings coming from living cells is that, in this case, nature also gives us plenty of tools (enzymes, recognizing systems, interacting systems) to process them and to perform molecular computations. Moreover, the biomolecular world is well known, and it may prove more interesting because of the numerous applications to medicine, thus fascinating us because of the influence on, and the proximity to, our own life. It turns out that, in this context, DNA polymers are usually favorite objects of interest.

As matter of fact, DNA evolved to become the primary carrier of genetic information thanks to its extraordinary chemical properties. The same properties also make DNA an excellent system for the study of biomolecular algorithms, including self-assembly, which is a wonderful example of self-organization process [155]. Intra-molecular interactions between DNA single strings can be precisely predicted by Watson-Crick base pairing, and these interactions are structurally understood at the atomic level. Given the diversity of DNA sequences, one can easily engineer a large number of pairs of DNA molecules that associate with each other in specified sequences and in a well-defined fashion. This property is not common with other molecular systems. Small organic and inorganic molecular pairs can interact with each other with specificity and in well-defined structures, but the number of such pairs is limited and their chemistry varies greatly. Protein molecules, such as antibody-antigen pairs, have great diversity and high specificity [11]. However, it is extremely difficult, if not impossible, to predict how do proteins interact with each other. Moreover, DNA sequences have the advantage to be stable (a dry DNA molecule has a stability of about one billion of years!) and easily manipulable even

[4] The author thanks her advisor for this stimulating question.

at room temperature, they have convenient modifying enzymes and automated chemistry, they may be heated without suffering any damage and, when paired, they form an externally readable code [5].

PNA is an artificial analogue of DNA [107]. It consists of a sequence of four bases, like DNA, along with a peptide backbone, like proteins. PNA can hybridize nucleic acids by hydrogen bonds of complementary bases and has interesting distinct features. Arbitrary short sequences can be synthesized like DNA oligomers, but PNA has higher sequence specificity than other nucleic acids; in fact, the PNA-DNA complex has higher melting temperature than the DNA-DNA one. This fact makes it almost irreversible the phenomenon of strand displacement, where PNA displaces itself into a double strand of DNA by taking over the hydrogen bonds. This ability allows one to form triple helices in a totally irreversible manner. Moreover, the PNA peptide backbone has no electric charge, thus no standard electrophoresis can be carried out for length-based selection. Irreversible formations by PNA have been useful in aqueous computing for write-once operations [241], and operations combining PNA and DNA were used in ad hoc algorithms [196, 203]. Although the ability of PNAs to bind to complementary single string DNA with extremely high affinity and sequence specificity is well characterized, the feasibility of DNA computing still is the most intriguing challenge for the design of string based bio-algorithms.

1.2 DNA computing

DNA computing is an emerging area of computer science where information is stored in form of DNA polymers, that are manipulated by enzymes in a massively parallel way, according to strategies that can produce computationally universal operations [185, 79].

Since the discovery of the DNA structure over fifty years ago[5], extraordinary advances in genetics and in biotechnology have allowed the development of a kind of research which employs this molecule of life as a nanomaterial for computation. The laboratory possibilities involving DNA molecules have offered a fresh paradigm for performing and viewing operations, and several models for DNA-based computations have been developed, including *insertion-deletion systems*, *splicing (finite, linear, circular, distributed) systems*, *sticker systems*, *molecular finite automata*, and *microfluidic reactors*.

[5] In 1962 James Watson, Francis Crick, and Maurice Wilkins jointly received the Nobel Prize in medicine or physiology for their discoveries concerning the molecular structure of nucleic acids and its significance for information transfer in living material (Nature 171, pp 964-967, 1953). Since the Nobel Prize may only be awarded to living persons, Wilkins' colleague Rosalind Franklin, who had died of cancer in 1958 at the age of 37, could not be honored. Their work followed Linus Pauling's discovery that the molecules of some proteins have helical shapes (Nature 161, pp 707-709, 1948), and Erwin Chargaff's experimental finding that there are as many A as T bases and as many G as C bases in DNA (Experientia 6, pp 201-209, 1950). Their great discovery was also inspired by the book *What is life? The Physical Aspect of the Living Cell*, where Erwin Schrödinger had guessed a polymeric structure of DNA through only theoretical speculations (1944).

Insertion-deletion systems were first considered by Gheorghe Păun in [177], and then by other authors, for example in [126]. They are computational systems based on context-sensitive insertion and deletion of symbols or strings. They have been studied from a formal language theory viewpoint, initially with motivations from linguistics and then with inspirations from genetics, since both inserting and deleting may be easily seen as operations performed by enzyme action or by mismatching annealing [185]. An experimental counterpart of such systems could be the programmed mutagenesis, implemented in [131]. This is a sequence-specific rewriting of DNA molecules, where copies of template molecules are produced that feature engineered mutations at sequence-specific locations.

The first achievement by mathematical analysis of biochemical operations of DNA recombination had resulted in the formulation of splicing systems, introduced by Tom Head in [97]. Roughly speaking, a splicing operation is a new type of string rewriting rule on double DNA sequences, inspired by a phenomenon of DNA recombination in vivo which involves restriction enzymes (recognizing a specific 'site' and cutting the sequence at that position) and ligase enzymes (pasting two molecular fragments by means of their matching ends). Splicing systems were proposed to model the mathematical (linguistic) properties of DNA recombinant behaviour, namely in the paper [98] Tom Head suggested the use of circular (cyclic) strands within splicing systems, but a few years later the H systems were considered as a theoretical model of DNA based-computation.

Extended splicing models have been proposed to the purpose of enhancing the computing capability beyond the regularity bound [146], in the hope to achieve Turing machine universal computability [179], while staying within the realistic framework of only allowing finite sets of axioms and of rules. In [184], for example, Gheorghe Păun, Grzegorz Rozenberg and Arto Salomaa considered splicing rules controlled by a next-rule mapping, like in programmed grammars, and found an abstract model where the splicing rules are themselves modified from step to step by means of pointwise mutations of the one-symbol insertion-deletion type. Although extended H systems with regular sets of rules characterize the family of recursively enumerable (RE) languages, yet as long as they have finite sets of axioms and of rules their language generating power cannot exceed the family of regular languages [147]. Quite surprising in this context is the result by Matteo Cavaliere, Nataša Jonoska and Peter Leupold, who obtained universal computation by using finite splicing systems equipped with simple observing and accepting devices that consist of finite state automata [38]. More details on simple H systems may be found in Section 3.2.

Splicing systems were extended to tree structures in [208], and the language family generated by tree splicing systems was shown to be the family of context-free languages. Universal models based on H systems with permitting and forbidding context, inspired by in vivo promoters and inhibitors respectively, were investigated as well [179]. Finally, splicing membrane systems concerning membrane computing with string objects and splicing rules were investigated [178], and structures that are similar to parallel communicating grammar systems were obtained in the form of distributed H systems [47, 49], where different parts of the model work independently and the computation result is deduced from the partial results produced in

such parts. With three context-free components, one can characterize the family of RE languages.

Regular simple sticker systems have been introduced in [124], where primitive balanced and fair computations of dominoes were discussed. They were then extended into bidirectional sticker systems, where the prolongation in both directions was conceived, so that they appeared in a general form with dominoes of arbitrary shapes corresponding to DNA molecules [182]. A classification of sticker systems as well as their generative capacity was thoroughly investigated in [185].

The automata counterpart of the sticker systems are the Watson-Crick automata, introduced in [80], which handle data structures different to the customary ones. Their objects are in fact double strings assumed to be paired by the Watson-Crick complementarity relation, rather than linear, one-dimensional strings of symbols, and automata that (at least in their initial definition) scan each of the two strands separately, albeit in a correlated manner. Although these automata exploit a DNA-like support for computation, they perform operations in the standard sequential way, thus lacking the massive parallelism of DNA computation.

A recent breakthrough in the approaches to using DNA as an autonomous machine for computation was achieved in Israel by a group leaded by Ehud Shapiro [14]. The combination of a type of restriction endonuclease enzyme called FokI, used as a tool to change states in a finite state automaton, and a rather smart encoding of states and symbols, yielded a successful in vitro implementation of two-state DNA automata. Three main ideas were exploited: use of the restriction endonuclease FokI, encoding (state, symbol) pairs with both sequence and length of the segment, and consideration of accepting sequences for final readout. Such a DNA automaton changes its states and reads the input without outside mediation, solely by the use of the enzyme. The authors demonstrated simulations for several automata with two states. In the implementation, a total amount of 10^{12} automata sharing the same software ('transition molecules') run independently and in parallel on inputs in 120 μl solution, at room temperature at a combined rate of 10^9 transitions per second with a transition fidelity greater than 99.8 %. However, this method encounters certain limitations when trying to extend it to more than two states.

Takashi Yokomori, Yasubumi Sakakibara and Satoshi Kobayashi, in [244], have proposed a theoretical framework, using a length-encoding technique, that presents no limitations in implementing nondeterministic finite automata with any number of states. In order to encode each state transition with DNA single strands, no special design technique for the encoding is required, but only the length of each sequence involved is important. A computation process is accomplished by self-assembly of encoded complementary strands, and the acceptance of an input string is determined by the detection of a completely hybridized DNA double strand. Experiments in vitro to implement such a model are reported in [135]. Finite automata ranging from two to six states were tested under several inputs, and in particular one six-state finite automaton executed its computation, recognizing a given input string. The result is definitely noteworthy, though a generalization would probably present experimental difficulties. Indeed, the imprecise biotechniques (like extraction by affinity purification) employed in the implemen-

tation risk to jeopardize the result, and so also do the repeated patterns in DNA sequences, that produce several unexpected, redundant PCR products (material dissipation).

Finally, a (theoretical) way to implement general computing devices with unbounded memory was suggested in [39], where one may find a procedure to implement automata with unbounded stack memory, push-down automata, by using circular DNA molecules and a type of restriction enzyme called PsrI. The same ideas are there extended to show a way to implement push-down automata with two stacks (i.e., universal computing devices), by using two circular molecules glued together by means of a DX molecule, and a class of restriction enzymes.

The research related to micro-flow reactors has started in [89] and has the goal to improve the programmability of DNA-based computers. Novel clockable microreactors are connected in various ways, and DNA strings flow through channels where programmable operations can be performed [228]. Compared to test tubes, microfluidics may have certain advantages, such as small amount of solution, high reaction speeds, and the possibility of easy automation [229].

1.2.1 Theory and experimentation

Here I would like to portray how the flavour of DNA computing models is changing during the latest years. Initially they were pretty theoretical, often their implementability was just hypothetical, and usually it was assumed. Models such as sticker and splicing systems were widely investigated from a formal language theory point of view, and gave us an articulated and abstract understanding of DNA behaviour. A nice example of this tendency is the paper [152], where the authors proposed the syntagm 'DNA computing in info', as opposed to both in vivo and in vitro, and stressing the fact that they worked in an idealized, mathematical framework. Actually, they sharply showed that, the abstraction of a DNA computing strategy, which was used in several algorithms (for example, in [203]) and experiments, enables us to compute beyond Turing power, by using devices (finite state machines) much weaker than Turing machines. Their mathematical assumptions are comparable to the passage from computers to Turing machines-endowed with infinite tapes and working an unbounded number of steps, and the claim is simply deduced from the fact that the family of RE languages is not closed under complementation.

Autonomous DNA automata and micro-flow reactors, instead, are meant to become really implemented as computational devices. The implementation of finite automata with few states is not particularly interesting from the theoretical viewpoint of formal languages, but it represents a significant advance towards the ultimate goal of achieving a biomolecular computer. It has shown that, with proper coding and use of appropriate enzymes, a computational device is possible without any outside mediation. This opens up the door to using such devices not just to perform computation, but potentially also in genetics and in medicine.

As it were, the formulation of theoretical models has spurred the production of actual implementations, favoured by continuous improvements in the underlying technologies; yet no implementation deserves interest *per se*, rather as long as a theoretical model exists that can provide it with foundations, as well as prove,

generalize, or just confirm the results it yields. The highly and genuinely interdisciplinary field of molecular computing is an evident example where theory and practice grow up together, just because of their interchange and complementarity, in tight duality, as facets of the same coin [5]. Namely, self-assembly of DNA molecules has recently inspired models that combine theoretical and experimental aspects. Let us start with a few basic principles of DNA self-assembly.

Two complementary molecules of single-stranded DNA have paired bases that bind to each other and form the well known double helix structure (duplex). A linear molecule can fold on itself, in that case it forms a hairpin structure, which is provoked by autonomous hybridization between two complementary subsequences of the single stranded molecule (for a picture see Figure 3.13, which is taken from [73]). Two molecules of double-stranded DNA (duplexes) can further glue together if they have complementary single-stranded overhangs (sticky ends)—for a picture see left side of Figure 7.7 coming from [73]. A 3-arm branched molecule consists of three duplex arms arranged around a center point (for a picture see Figure 7.8, which is taken from [74]), while a double crossover molecule (DX) consists of two adjacent duplexes with two points of strand exchange. Formal language characterizations of the computing power of molecular structures (with more attention to hairpin formations) are explored in [186].

Erik Winfree pioneered a thorough investigation on models of self-assembly [239, 235]. The physical system considered from a formal language theory viewpoint in [239] was the following. We synthesize several sequences of DNA, mix them together in solution, heat it up and slowly cool it down, allowing complexes of DNA to form chemically or enzymatically ligate adjacent strands. By denaturing the DNA again, in the solution we find new single stranded DNA sequences formed by a concatenation of the initial ones. In other words, some languages of $\{A, T, C, G\}^\star$ are generated by such a self-assembly system. The result proved in [239] is that the family obtained by admitting i) only duplex formations is the family of regular languages, ii) only duplex, hairpins, and 3-armed junctions is the family of context-free languages, iii) only duplex and DX molecules is the family of recursively enumerable languages. Moreover, in [235], Erik Winfree proposed the DNA block model, and together with Tony Eng and Grzegorz Rozenberg proved that the self-assembly of DNA tiles (also called two-dimensional self-assembly) is capable of simulating a universal Turing machine [237], by showing that the four sticky ends in a DX molecule can be considered as four sides of a Wang tile[6]. By tiling the plane with Wang tiles one can simulate the dynamics of one-dimensional cellular automata, whence that of a Turing machine.

From the experimental viewpoint, nanoscale guided constructions of complex structures are one of the key challenges involving science and technology in the twenty-first century. One of the first synthesized DNA molecules with a structure deviant from the standard double helix is the stable four-junction molecule (now known as J1), that was designed in the late 80's in the laboratory of N.C.

[6] A Wang tile is a planar unit square whose edges are colored. A tile set is a finite set of Wang tiles. A configuration consists of tiles which are placed on a two-dimensional infinite grid. A tiling is a configuration in which two juxtaposed tiles have the same color on their common border. These concepts were introduced in: H. Wang. Proving theorems by pattern recognition II. *Bell Systems Technical Journal*, 40: 1-41, 1961.

Seeman [215]. This molecule now is one of the basic building blocks used for design and assembly of various different constructions in the rapidly growing field of DNA nanotechnology. It has been used as a basis for more complex building blocks, consisting of double and triple cross-over molecules [234, 136], as well as for junction molecules with more than four branches [230]. These arm branched molecules are employed in a construction of two-dimensional arrays [158], and are suggested for growing a DNA fractal-like molecule [33], for assembling arbitrary three-dimensional graphs [206], and even for obtaining DNA Borromean rings [157] (see Figure 1.1, where the picture on the left side is taken from [157] and the picture on the rigth side is taken from [118]).

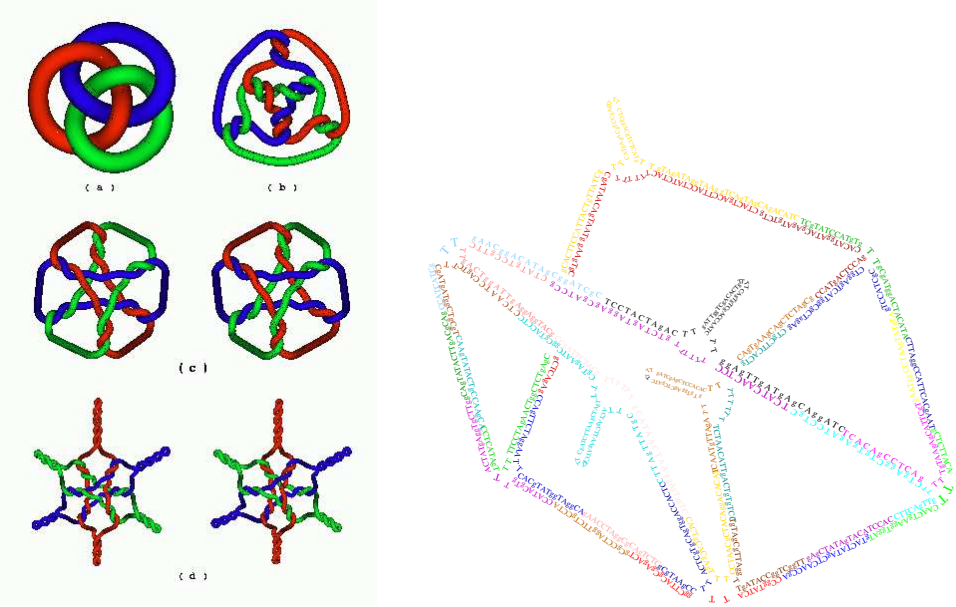


Fig. 1.1. Borromean rings and irregular graphs.

Two- and three-dimensional DNA assemblies have been suggested and demonstrated for information processing [239]. Experimental demonstrations, for some of these ideas, were obtained through the construction of the Sierpinski triangle [199] as well as by the linear assembly of TX molecules (triple cross-over molecule) encoding a XOR computation [156]. Very recently, a simple method of "origami" has been introduced[7] for folding long, single-stranded DNA molecules into arbitrary two-dimensional shapes by short staple strands. In fact, this one-pot method uses a few hundred short DNA strands to 'staple' a very long strand extracted from a virus, into two-dimensional structures that adopt any desired shape, included smiling faces! While the use of branched junction molecules for computation through

[7] By P.W.K. Rothemund, in the paper "Folding DNA to Create Nanoscales Shapes and Patterns", Nature 440, pages 297-302, 16 March 2006.

assembling three-dimensional structures was suggested in [118], by demonstrating how NP-complete problems can be solved by one-step assembly.

1.2.2 Attacking NP-complete problems

One of the first aims of DNA computing was the solving of NP-complete problems, since the massive parallelism and the intrinsic nondeterminism make hard problems tractable, even by using brute force search of solutions ("Molecular computing is justifiably exciting, not least for the alluring prospect of biologically-inspired machines nicely handling NP-complete problems", T. Kazic, [130]). However, the scaling up of the proposed models from toy instances to realistic size remains an open problem, mainly because of the increasing quantity of DNA strands necessary to encode the initial data. For a graph having the modest size of two hundred vertices, with the Adleman's strategy one would need "DNA material more than the weight of the Earth" (J. Hartmanis, [96]). For this reason, a few algorithms were proposed where the potential solutions are not constructed all at once [169,245,118]. The use of linear duplex molecules to solve NP-complete problems has been surperseded by self-assembly of DNA tiles [199], and more generally by graph self-assembly with junction molecules [230].

The potential of using DNA molecules for solving computationally hard problems has been extensively investigated both from experimental and from theoretical points of view. The satisfiability problem (SAT) was perhaps the most widely studied NP-complete problem, tricky theoretical solutions of which can be found in [112, 245, 154, 203], while experimental solutions of small instances of SAT are proposed in [143, 210, 209, 29, 28]. An example of a (theoretical) algorithm in vivo solving SAT can be found in [60], where the boolean circuit is genetically encoded into DNA which is ideally inserted into a cell, and the cell is programmed to select a random assignment of variables, test for satisfiability, and display a suitable phenotype when a satisfying assignment is chosen. A theoretical solution of the road coloring problem is presented in [110], while an experimental solution of a small instance of the maximal clique problem is reported in [175]. DNA has been used to solve the knapsack problem [188], and even to implement a simplified version of poker [240].

After Adleman's experiment where a solution of an instance of Hamiltonian Path Problem was found through DNA strands [1], Lipton showed that the satisfiability problem can be solved using essentially the same bio-techniques [143]. The schema of the Adleman-Lipton *extract model* consists of two main steps: the combinatorial *generation* of the solution space (an initial pool containing DNA strands encoding all the *possible* solutions) and the *extraction* of (the strands encoding) the *true* solutions. Serious limitations about the quantity of DNA necessary to encode the initial pool of the extract model came up soon [96]. Indeed, it should be enough to guarantee the presence of all possible solutions probabilistically, and that requires an amount of DNA that is (at least) exponential with the size of the instance. Nonetheless, implementation of algorithms solving (toy) combinatorial problems by the extract model is often exploited to improve generation and extraction methods, that are fundamental operations in biology and in genetic recombination. On the other hand, various enhancements and alternative approaches to the extract model were suggested.

By way of example, we recall a few algorithms that solve the satisfiability problem by skipping the error prone, laborious and expensive extraction phase of Adleman's model. They are based on hairpin formation [210, 209, 94], on a length-based method [109], and on mathematical properties of specific sets of natural numbers [84]. Namely, in [209] hairpin formations represented inconsistent assignments of variables and, once detected, they were destroyed by digestion of an enzyme. Indeed, each string represented a random assignment, where opposite boolean values were encoded by complementary substrings. In the length-based method [109], costs of paths (in the weighed graph problem) are encoded by the length of the oligomers in a proportional way. The advantage is that, after an initial pool generation and amplification, a simple gel-electrophoresis can separate the respective DNA duplexes according to their length, which directly decodes the solutions. In [84] the solution can be uniquely detected by length as well, and the trick lies in the encoding. This was inspired by [154], and it is based on the nice characteristic property of unique-sum sets of natural numbers which is defined as follows. Set S is *unique-sum* if the only way of obtaining the sum, say s, of its elements as the sum over a multiset M of elements from S, is by taking each of them only once, that is, with $M = S$. Clauses are encoded by oligomers whose lengths are given by numbers out of a unique-sum set S, so that, when the SAT instance is suitably encoded by a graph, a directed path starting from the initial node and ending up to the final one represents a solution if, and only if, it is of length s.

Standard bio-operations were also used to break the Data Encryption Standard (DES) [26], one of the cryptographic systems in widest use previously. Approximately one gram of DNA was needed and, by using robotic arms, the breaking of DES was estimated to take five days. A most significant aside of the analysis for breaking DES, was that the success is quite likely even with a large number of errors within the laboratory protocols. The DES breaking problem is still used as an example of application of alternative, biologically inspired methods [6].

The theoretical storage potential of DNA molecules is impressive—one gram of dry DNA occupies a volume of approximately one cubic centimeter and it can store as much information as approximately $2\ 10^{21}$ bits—when compared to the storage capacity of current technology, that is approximately 10^9 bits per gram. As a matter of fact, DNA molecules approximately have an information density of 2 bits per cubic nanometer versus 1 bit per 10^{12} nm^3 for current technology. Yet, the actual capacity of DNA oligonucleotides of a given length to store information in the globally robust and fault-tolerant way that biology requires, seems to be very difficult to quantify, or even to estimate [87, 189]. Some studies explored the use of DNA databases for storage of information and for its retrieval by search techniques, based on query output by bead separation using a fluorescence activated cell sorter [13, 193]. Test databases of increasing diversity were indeed synthesized and the construction of an extremely large database was completed. Capacity, data security and specificity of chemical reactions were improved through some experiments, for example in [129, 40]. The way cells and biomaterials store information is one of the most important and enigmatic problems of our times. Encoding information in biomolecules for processing in vitro has indeed proven to be a challenging problem for biomolecular computing. Codeword design and, more generally, data

and information representation in DNA receives an increasing interest, not only in order to use biomolecules for computation, but also to shed light on a number of other problems in areas such as bio-informatics, genetics and microbiology.

1.2.3 Emerging trends

The code word design problem requires producing sets of strands that are likely to bind in desirable hybridizations, while keeping to a minimum the probability of erroneous hybridizations (such as cross-hybridization of non-complementary strands, or formation of secondary structures) that may induce false positive or false negative outcomes. It turned out that this is quite a complex problem, and several authors have concentrated their studies on developing theoretical coding models [106, 114, 113] (see also Section 3.5), computer simulations [86, 214], and even experimental builds of coding libraries [50]. Usually, word sets are carefully designed by using computational and information-theoretic methods, so that the resulting long strands behave well with respect to computation. Direct encoding is, however, not very efficient for the storage or processing of massive amounts of data, because of the enormous implicit cost of DNA synthesis to produce the encoding strands. Indirect and more efficient methods have been proposed in [87] using a set of non-cross-hybridizing DNA molecules as basic elements. Therefore, generation of DNA libraries is a crucial point for the effective production of DNA memories, and the improvement of generation methods appears highly relevant.

Unfortunately, as a matter of fact, several standard biomolecular protocols do not ensure the needed precision for computation. Such techniques have to go through several adjustments, and sometimes completely new protocols are necessary in order to improve their yield. The solution of a 20-variable SAT problem in [28], for example, was obtained by designing protocols for exquisitely sensitive and error-resistant separation of a small set of molecules. Hence, the search for a DNA solution of such combinatorial problems, even though not computationally significant, may prove fruitful in developing new technologies [111]. An attractive task nowadays is indeed to characterize models within the scope of the given experimental limitations, and to improve or obtain protocols that are sufficiently reliable, controllable and predictable. This thesis is mostly along this trend, and it has been exciting to see how the algorithmic analysis of a laboratory procedure (namely PCR) may suggest new techniques with useful applications. It is convenient to formulate methods of molecular biology as biomolecular algorithms, and to study the corresponding molecular processes both from combinatoric and experimental points of view. The goal is to optimize the efficiency of biomolecular methodologies, not only to improve the reliability of computing, but also for medical and biological applications ("Optimization methods should be viewed not as vehicles for solving a problem, but for proposing a plausible hypothesis to be confirmed or disconfirmed by further experiments", R. M. Karp, [127]).

A first step towards a better expertise in DNA manipulation is the understanding of DNA spontaneous behaviour, namely by means of algorithmic and computational analysis of its combinatorial aspects. In this respect, the study of gene assembly in ciliates is an emergent research line of DNA computing [57, 190], discovering and modelling biomolecular processes that take place in living cells.

Ciliates are indeed unicellular eukariotes (in particular bi-nucleated cells), named after their wisp-like covering of cilia, that perform one of the most complex examples of DNA processing known in any organism, the process of gene assembly.

In their micronucleus, coding regions of DNA (named MDS as 'macronuclear destined sequences') are dispersed individually or in groups over the long chromosome, along which they are separated, and in a sense obscured, by the presence of non-coding DNA sequences (named IES as 'internally eliminated sequences'). During sexual reproduction, a macronucleus develops from a micronucleus, through a sophisticated rearrangement mechanism called gene assembly, that excises IESs and orders MDSs in a contiguous manner, to form transcriptionally competent macronuclear genes. Unscrambling is a particular type of IES removal, where the order of MDSs in the micronucleus is often radically different from the one in the macronucleus. Discovering which are the assembly strategies of ciliates for a given micronuclear gene has been a truly fascinating computational task.

Three types of formal systems were investigated at various levels of abstraction: MDS descriptor, string pointer, and graph pointer reduction systems, and as far as assembly strategies are concerned they turned out to be equivalent [54]. Three intramolecular operations ('ld', 'hi', and 'dlad') have been postulated for the gene assembly process, and they provided a uniform explanation for all known experimental data. The gene structure and the operations themselves have been modeled at three levels of abstraction, respectively based on permutations, strings, and graphs [56]. The computational power of molecular operations used in gene assembly was previously investigated for a model with intra- and inter-molecular operations in [139]. Meanwhile, more recently, a model was proposed by David Prescott, Andrzej Ehrenfeucht and Grzegorz Rozenberg, that is based on recombination of DNA strands guided by templates [191], and its computational power was studied as well [48].

I like to conclude this section with words of an expert in the field, Nataša Jonoska: "Whatever the results, this area of research has brought together theoreticians (mathematicians and computer scientists) with experimentalists (molecular biologists an biochemists) to a very successful collaboration. Just the exchange of fresh ideas and discussions among these communities brings excitement, and quite often, provides a new line of development that could not have been possible without the 'outsiders' " [111] .

1.3 This thesis

Theoretical computer science studies the basic mathematical properties and limits of computation, such as computability (what can we compute?) and complexity (how fast can we compute?), by investigating models of computation, such as artificially built models and natural phenomena. The research area of molecular computing where this thesis finds its place, lives in this context, with major adherence to DNA computing.

Molecular computing assumes an abstract notion of molecule as a floating entity in a fluid environment. DNA and membrane computing investigate two different aspects of this idea or, more precisely, the molecule behaviour is analyzed from two

different observation levels. In the context of DNA computing the molecule is a sequence with recombinant behaviour, while in the context of membrane computing it is an element which transforms and moves from one compartment to another of a cellular-like system. Of course, the two aspects are linked, and in nature they correspond to the interactions between nucleus and cytoplasm of eukariotic cells. These are complementary perspectives risen from a common inspiration, hence in Chapter 2 we present both of them. A presentation of the DNA structure is indeed followed by an excursus in the membrane computing world together with an application of membrane systems to an immunological process.

We start from the concept of string, by presenting a few investigations about the structure and the interactions of DNA sequences, which are *floating* strings. Bilinearity, complementarity and antiparallelism, typical of the double stranded DNA structure, are proven in an abstract setting, as efficiency requirements on algorithms duplicating 'mobile strings'. Such duplication of strings is then extended to the following operations on numbers encoded within strings: duplication, triplication and division by six. These operations guarantee the universality of a DNA-like computational system based on rules that have a biotechnological interpretation.

As an attempt at looking at our floating strings from a more abstract level, where the strings lose their own structure and become objects moving between compartments, we present membrane systems along with an application to describe the selective recruitment of leukocytes. Actually, membrane computing has been mainly investigated from two different viewpoints: a model as realistic as possible according to the bio-reality, and a computational device as powerful, elegant and efficient as possible. We worked along the first research line; while the second one studies the equivalence of P systems to Turing machines, even with reduced membranes, rules, objects, and explores their potential to solve hard problems. In both cases, the cell biochemical system is redesigned so that it more or less closely comes to fulfil the assumptions of the theory of computing, and then it is examined how well does the theory predict its properties, whether computational or biological ones.

In our speculations on DNA computations, where theoretical issues were supported by experiments, we realized that a special kind of PCR, that we call *Cross Pairing PCR* or shortly XPCR, can be the basis for new algorithms that solve a wide class of DNA extraction and recombination problems. In the middle chapters we describe such a technique as a biotechnological tool to extract strings containing a given substring from a heterogeneous pool, to generate DNA libraries, and to perform mutagenesis. More in detail, the following three tasks are addressed.

1. Generating a combinatorial library of n-bit binary numbers, encoded in DNA strands $X_1, Y_1, \ldots, X_n, Y_n$, that is, producing the following pool, containing 2^n different kinds of strings:
$$\{\alpha_1 \cdots \alpha_n \mid \alpha_i \in \{X_i, Y_i\},\ i = 1, \ldots, n\}.$$
2. Extracting all those strings from a given pool P which include a given substring γ, that is, obtaining a pool that consists of all the strings out of P that contain γ as a substring:
$$\{\alpha\gamma\beta \mid \alpha\gamma\beta \in P\}.$$

3. Performing mutagenesis(X, Y) on a pool P, which means substituting every occurrence of X with an occurrence of Y over the strings of P, that is, turning it into the pool:
$$\{\alpha Y\beta \mid \alpha X\beta \in P\}.$$

A few bio-algorithms are proposed, mainly based on polymerase action, that have noteworthy advantages, such as efficiency and feasibility, with respect to methods from the literature. Mathematical analysis as well as laboratory experiments are presented as supporting our work. This includes a study that is linked to splicing systems, since XPCR suggests a new way to implement a special kind of recombination that is performed in nature by only few enzymes (about thirty). We designed and implemented an algorithm, that produces a DNA library as the strictly locally testable language generated by a simple splicing system [164]. From a theoretical point of view, splicing systems started with this model, while an algebraic characterization of this type of splicing systems was obtained in [103], the first one for any class of splicing systems. From a biological point of view, this result is encouraging, in that it reveals a new capacity to manipulate DNA for genetic recombination and for the creation of a clone library (that is the starting point of any physical mapping effort in order to sequence a genome).

The aforementioned technique was applied to extraction and mutagenesis algorithms as well. The first application proves interesting in DNA computing, since the extraction operation performed by standard protocols is still problematic and error prone, while both applications seem to enjoy a biological relevance, for example in the study of genetic mutations.

The powerful molecular recognition of Watson-Crick complementarity employed in DNA base pairing is also used in various models of biomolecular computing and information processing, to guide the assembly of complex DNA structures. In particular, the process where substructures are spontaneously self-ordered into superstructures, that is driven by selective affinity of the substructures, is called self-assembly. In Chapter 7 we present a particular family of graphs as a theoretical model for the generation of self-assembled DNA forms. The model is a variant of the forbidding-enforcing systems, introduced in [55] as a model of chemical reactions. We extend this original model to the construction of three-dimensional structures and, in particular, we concentrate on structures obtained by DNA self-assembly. On the other hand, as a systematic way of describing classes of graphs, our model may be considered as a starting point for developing new ways to investigate graphs in classical graph theory [74].

In conclusion, the thesis presents both theoretical models of molecular computations and novel DNA-based algorithms, corresponding to procedures that are relevant in biological contexts, and bringing the experimental nature of DNA computations to the fore. Namely, specific methods for DNA extraction [72], recombination [77], and mutagenesis have been investigated. A schematic overview follows, that summarizes the contributions of this thesis with respect to the topics outlined in the previous sections.

In the context of theoretical models, we propose:

- A membrane model for leukocyte selective recruitment [75, 43](collaboration with Vincenzo Manca, Dep't of Computer Science, University of Verona, Italy).

- A DNA register machine (collaboration with Maurice Margenstern, LITA, University of Metz, France; work inspired by ideas from the paper [76]).

- A few forbidding-enforcing conditions on graphs, that describe self-assembly processes [74, 73] (collaboration with Nataša Jonoska, Dep't of Mathematics, University of South Florida, USA).

In the context of combinatorial and algorithmic investigations on novel DNA manipulation methods, we put forth (collaboration with Vincenzo Manca):

- the XPCR technique
- an XPCR Extraction Algorithm [72]
- an XPCR Mutagenesis Algorithm [150]

and, with more autonomous work of mine:

- an XPCR Recombination Algorithm [77, 68]
- an Enzyme-based Extraction Algorithm [71]

Finally, we exhibit three experiments which were carried out to test all XPCR-based algorithms (collaboration with Vincenzo Manca, together with Cinzia Giagulli and Carlo Laudanna from the Dep't of Pathology, University of Verona, Italy).

The work of interaction, filtering and translating between the theoretical and the experimental sides of this research has been the most challenging part. Although it is not an 'orthodox' computer science research, we think it has been precious from a scientific point of view, even exciting, because it highlights an experimental nature of strings and algorithms.

1.4 Conclusive notes

I should like to cite some relevant and comprehensive reference books where all basic concepts we are dealing with may be found. A standard and very good textbook on molecular biology is [2], whereas the subjects of DNA computing and membrane computing can be both approached and deeply understood by reading [185, 5] and [180, 43], respectively. Molecular computing bibliography increases continuously, though. Main results and open questions about membrane computing can be found at the website http://psystems.disco.unimib.it, whereas the proceedings of the annual workshop on membrane computing (currently at its sixth edition) are the best ongoing reference for following the vivid research production in this field (WMC7 will be at Universiteit Leiden, in Leiden, The Netherlands, on July 17-21, 2006). Both the website and the workshop are organized under the auspices of the European Molecular Computing Consortium, EMCC. A pretty good reference for the most up-to-date results, developments and main interests of DNA computing all over the world are the proceedings of the annual meetings

on DNA-based computers, that started in 1995 and are currently at their eleventh edition (DNA12 will be at Seoul National University, in Seoul, South Korea, on June 5-9, 2006). Finally, as a reference book on Formal Language Theory it is definitely suggested [202], whose notation we adopt in this thesis.

Since one of the goals of this chapter is to introduce molecular computing, and especially DNA computing, to as many people as possible, a few answers to frequent questions I was asked by inexperts in the field follow. With the same intent, those who want to just get a general idea about the subjects of this introduction are advised to consult the following recent, comprehensive articles: [243] is a long and detailed manuscript on molecular computing; [111] is a nice to read survey of some concepts and developments in DNA computing; [181] is an intuitive and interesting introduction to membrane computing, about which one may find lots of details in [183].

While attending conferences on different topics, computer scientists often asked me: What do you mean by 'complexity'? Can biocomputing relieve us from complexity? Isn't hardness an intrinsic feature of the problem? Why was the biggest SAT solving experiment carried out on 20 variables rather than directly on significantly sized instances? How precise are such computations?

Although there exists no well-founded definition of 'biocomplexity', yet this is clearly something different from classical complexity [5]. After an initial attempt to characterize complexity of DNA-based algorithms and protocols in terms of the traditional concepts of time and space, a notion of complexity seemed necessary that could capture other aspects of the physical-chemical reality (volume, temperature, amount of DNA and nucleotides) wherein biocomputation takes place [88]. A possible definition considers the number of different sequences as 'space' complexity of a test tube of DNA [62], and the 'time' complexity of an algorithm as the number of 'biosteps' (elementary physical or chemical steps, such as the changing of temperature or the action of an enzyme), among which the extraction operations should be definitely considered the dominant ones. I should like to conclude that a little bit of experience and common sense is fairly important for estimating complexity of bio-algorithms.

A problem is called 'hard' when it belongs to the class NP, i.e., that of problems solvable in polynomial time by a nondeterministic Turing machine. Intuitively speaking, a problem S is NP-complete if every problem belonging to NP can be 'translated' in polynomial time into an instance of S. Hardness seems to be an intrinsic feature of a problem as far as we consider the Turing machine as reference model for every possible computation. Whenever we go to explore an alternative model, and especially a different way to compute, the hardness might disappear, and we would thus be 'saved from complexity'. This seems not to be the case with DNA computing though—as mentioned earlier, the initial hopes about solving NP problems have actually fallen beyond, while a different trend has came up, whereby combinatorial problems are just an expedient to understand natural phenomena and to improve our ability to manipulate biomolecules. In a DNA computing experiment, significantly sized problems still require a good deal of tasks that cannot be addressed by current technology and bioalgorithms. Research in DNA computing is pushing the barrier of problem size by suggesting and improving methods that may scale up better, but the achievement of solving

realistic problems is really far away. At the present infancy stage, techniques are just gradually analyzed, tested, and improved.

Although recombinant DNA operations are being automated and robotized, they are still not precise. Many computational steps are based on protocols which are designed to serve purposes that are particular to molecular biology (hence essentially different from those of DNA computing), and their precision depends too much on laboratory conditions that are generally unknown. In each laboratory protocol one assumes less than 100% yield, often less than 80%. As yield decreases with each additional protocol, it becomes valuable to avail of models and computational techniques that require little outside mediation or human interaction. Finding more precise methods, or improving the precision of current techniques, is indeed one of the research trends in DNA computing. In particular, the computation error rate is dominated by the error rates of the extract and combine steps [42].

Errors usually come from two sources: every operation may cause an error, and the DNA lives a half-life, that is, it decays at a finite rate. If the DNA algorithm execution takes months, for example, then there is a likely chance that the good strands disappear. An error may have huge consequences for biocomputing. The goal of bio-algorithms essentially is to keep good strands (encoding a solution of the problem) alive and to get rid of bad strands (encoding something else than solutions). If a good strand is damaged or lost, then the algorithm may fail. If bad strands are not removed, there may remain too many bad strands left at the end of the algorithm execution, and this also may cause failure of the algorithm. Right after Adleman's experiment, the problem of misclassification errors caused by faulty extracts and strand loss was discussed in detail [62, 88, 42]. Even without improvements in laboratory techniques, research has shown that solely algorithmic methods can reduce error rates significantly [128, 200]. In other words, every algorithm for a DNA computer can be mapped to an equivalent one that is comparatively error-free, admittedly at the cost of increased time and space.

More pragmatic questions used to come from curious friends: How much does a DNA computing experiment cost? Where does your DNA come from?

The cost of an experiment depends, of course, on the kind of experiment, besides the countries and companies where products are bought from, and it is hard to estimate an average cost. We worked mainly with PCRs and, as a gross indication, a kit of PCR products allowing one to perform 200 PCR reactions of $50\mu l$ of volume, costs around 200 euros. Actually, the most expensive ingredient turns out to be the polymerase. A good one, able to perform without errors for 1000 bases and also auto-correcting, costs up to 1000 euros per vial. The termocycler itself costs between 5000 and 10 000 euros (for real time PCR the cost goes up to 20000-30000 euros).

Sequences cost 0.80 euros per base. The kind of DNA that is employed in DNA computing experiments usually is synthetic, or "commercially prepared", including clones of human proteins or sequences extracted from human genes (that we used, for example, in the experiment reported in Section 6.3). Indeed, the DNA molecule is known at an atomic level, and it can be perfectly reproduced. The time required by current techniques to synthesize a DNA sequence of 100bp in average amounts to approximately ten hours.

2

Floating Strings

Nature uses only the longest threads to weave her patterns,
so that each small piece of her fabric
reveals the organization of the entire tapestry.

Richard P. Feynman[1]

DNA molecules have a structure that satisfies three basic principles: i) they are composed of two paired strings, ii) each symbol of a first string corresponds to its 'complementary' in the second string, iii) these strings are oriented along opposite directions [85]. These three principles are respectively referred to as: *bilinearity, complementarity* and *antiparallelism*. In Section 2.3, we show that they depend on a 'deep' logic that is dictated by some informational and computational aspects of the bilinear duplication algorithm described in Section 2.2. In other words, if one wants to design an efficient duplication system for strings (finite sequences of symbols) represented as mobile polymers (floating in a fluid environment), then one needs 'symbol molecules' asymmetric with respect to the three space directions (as the molecules are from a chemical point of view), and arranged according to the three principles mentioned above [76]. Moreover, some intrinsic features of DNA molecules, such as the geometry of the double helix, are proven to be implied by these principles [148], on which DNA processes are based on.

One of the (first) natural questions to ask about DNA computing concerns its potential to implement arithmetical operations [82]. In Section 2.4 a DNA-like system of rewriting rules on double strings is proposed. This is capable of universal computations, since it performs multiplication by two and by three, and division by six as well. It has been suggested as an extension of the bilinear duplication algorithm, and, it is computationally universal as a consequence of the (universality) results that one can find in [159].

Membrane systems are biologically inspired containers of floating strings. In Section 2.5 a general presentation of membrane systems is given, together with an application of them to analyze the immunological process of the selective recruit-

[1] The Character of Physical Law, 1965.

ment of leukocytes. The approach is that suggested by Gheorghe Păun in [181]: i) one examines a piece of reality, say a particular biological process, ii) one builds a P system modelling it, iii) one writes a program simulating the system[2] and performs experiments with that program, by turning certain parameters and looking for the evolution of the process.

2.1 DNA structure

Nucleotide bases, that are the building blocks of every piece of DNA, are usually represented by the letters A, C, G, and T, which stand for their chemical names: adenine, cytosine, guanine, and thymine. Yet in 1951 it was clear that the four bases are not present in the same amount in the organisms, and their relative proportion widely vary in organisms that are far in the evolution scale. Usually though, organisms have all the above four bases, except the virus *fago* T_2 (infecting bacteria) whose DNA has a modified version of the cytosine basis.

Nucleotides have three parts: phosphate group, sugar, and base. The carbons of sugar are numerated by $1'$ through $5'$, and those of the base by 1 through 5 (for biological details see [85] or [2]). The phosphate group hanging out of the $5'$ carbon can join with the hydroxyl group of another nucleotide, at the $3'$ carbon, to form a strong (covalent) bond known as phosphodiester bond. As it is displayed in Figure 2.1, which is taken from [85], through this concatenation one obtains one strand of DNA, i.e., a sequence of nucleotides bonded with phosphodiester bonds.

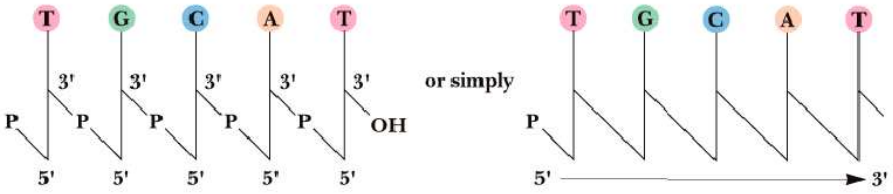


Fig. 2.1. Concatenation of nucleotides forming DNA strands.

The DNA strands have a natural orientation that is maintained by the concatenation through the phosphodiester bonds (see right side of Figure 2.1), while hydrogen bonds anneal two strands with opposite orientation. The A on one strand always pairs with the T on the other one, and C always pairs with G. Two complementary strands of DNA having opposite orientations thus joint together through hydrogen bonds, and form a double stranded DNA. The two strands are said to be Watson-Crick complementary to each other. In the following a double stranded DNA is briefly denoted with *dsDNA*.

DNA is found within living organisms in the well known double-helix structure, made of two strands wound up around each other. The two strands are antiparallel,

[2] In my case the program was written by Luca Bianco in [21].

that is, one strand runs 5′ to 3′ while the other one runs 3′ to 5′ (see Figure 2.2, which is taken from [85]). DNA is assembled in the 5′ to 3′ direction and, as a convention, we read it in the same way. The helix can be virtually of any length; when fully stretched, some DNA molecules are as much as 5 cm long. Nucleotides project in towards the axis of the helix, and each one forms hydrogen bonds with the one directly opposite it, forming base pairs, as displayed in Figure 2.2, which is taken from [69]. The pairing rules are such that if we read the sequence of nucleotides on one strand of DNA, we can immediately deduce the complementary sequence on the other strand. They are called the rules of Watson-Crick base pairing, named after the two scientists James Watson and Francis Crick discovered their structural basis in 1951 (for further details see footnote 5 of Chapter 1).

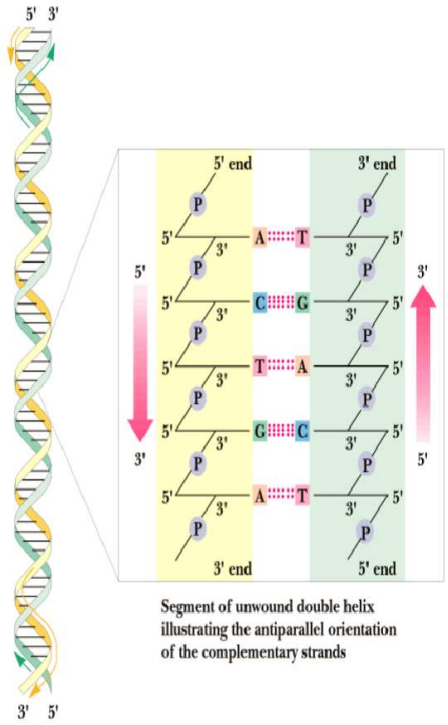


Fig. 2.2. Antiparallel orientation of DNA complementary strands.

One might wonder why adenine never bonds with cytosine, and thymine with guanine. Two purines (A and G) cannot fit within the helix (there is not enough space), while two pyrimidines (C and T) cannot get close enough to each other to form hydrogen bonds between them (there is too much space). However, the reason

is no longer space, but chemistry: only with the A-T and C-G pairs hydrogen bonds can take place[3], and with different strength [2].

Surprisingly enough, a double helix structure with identical angle and length proportions can be obtained as a consequence of purely geometrical argumentations as in [148]. The author has investigated the double stranded structure of DNA molecules in an abstract setting, where the principles on which DNA processes are based on were formulated in an axiomatic form. A mathematical analysis of interacting 'monomeric triangles' yielded to a deeper understanding of the logic internal to DNA structure, included the formation of major and minor groves in the double helix (see Figure 2.3, which is taken from [148]). This work was one of those cases where an algorithmic and computational investigation of natural processes can explain molecular structures and phenomena (see Section 1.2.3): "The possibility of figuring out mathematical DNA properties, by using general principles and abstract forms, seems to disclose new perspectives in the understanding of this marvelous biological reality" (V. Manca, [148]).

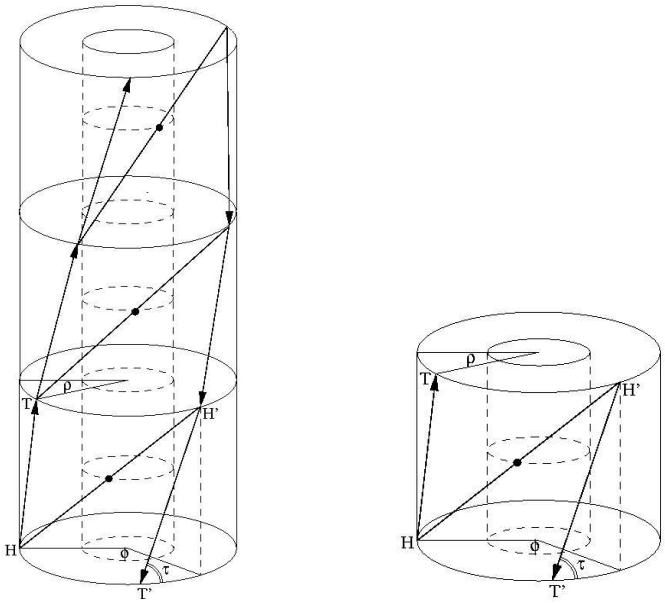


Fig. 2.3. Left side. Form resulting from monomers arranged in a bilinear non-planar structure. The whole structure is deduced from the values of three angles. Right side. The *rotation angle* ρ measures the rotation of the single monomer around the axis of the cylinder, the *phase angle* ϕ represents the rotation of the cylinder radius corresponding to two monomers that are paired, the *torsion angle* τ is the angle formed by the 'head-tail axis' of each monomer with the horizontal planes (orthogonal to the cylinder axis).

[3] I took this information and an inspiration for the description of the double helix from: R. Mosca, *Computational Methods for the Analysis of Genomic Data*, PhD thesis, Università degli Studi di Milano Bicocca, Milan, Italy, 2003.

The genome is the complete set of DNA contained in any cell of an organism, specifying how it will look like under a certain set of environmental conditions. It is duplicated whenever the cellular division occurs, and the double stranded molecule is copied inside the cell in such a way that each strand works as template for one of the strands to assembly, this process is said 'semiconservative' (see Figure 3.1, which is a composition of two pictures taken from [85]). The complementarity and the antiparallel orientation of the two linear components of DNA molecules are essential aspects of the logic underlying the autoduplication process [76].

In the following we use the expressions 'DNA strings' and 'DNA strands' indifferently, by neglecting the fact that the first term regards the formal languages where DNA is naturally formalized, and the second one the biological and physical context where it actually stays. Next section presents three possible algorithms duplicating formal strings: they work locally and they are presented as a grammar of a transition system with rewriting rules.

2.2 String duplication algorithms

Evolution works by making many copies of some organisms with few random changes and by selecting the 'best' ones. The process is repeated over and over again, and leads to a refinement of the species. In many works on artificial evolution, 'copying' is always taken for granted, being realized by centralized control program whose task is just to make copies of the genome. In nature it is not that easy though, since all organisms are responsible for their own copying.

Given a string α over an alphabet Δ we consider the problem of duplicating it and distinguish three different methods based on: i) transposition, ii) dissolution, and iii) pairing.

Duplication by transposition (TDA)

A transposition duplication algorithm (TDA) can be represented by the following set of replacing rules, where $\{a, b\}$ is the (terminal) alphabet, $i, j \in \{a, b\}$, and the other symbols are auxiliary (non-terminal) symbols. Given an input string α, let $S\alpha S'$ be an axiom of the system.

1. $Si \rightarrow X_i Y_i S$
2. $SiS' \rightarrow X_i{}' Y_i{}'$
3. $Y_i X_j \rightarrow X_j Y_i$
4. $Y_i X_j{}' \rightarrow X_j{}' Y_i$
5. $X_i X_j{}' \rightarrow X_i{}' j$
6. $Y_i Y_j{}' \rightarrow Y_i{}' j$
7. $X_i{}' \rightarrow i$
8. $Y_i{}' \rightarrow i$

Essentially, the linear structure is preserved and each symbol is equipped with a 'shadow' copy nearby it; after that, shadows are moved all together to one side of

the structure by keeping their relative positions, so that a shadow copy of the whole original string is obtained, whence a copy of the original string can be recovered[4].

For these replacing rules there is no explicit strategy, but if one wants to obtain strings composed of terminal symbols, then the primed symbols have to be transformed into terminals only at the end of the process.

Proposition 2.1. *Duplication by transposition performs a number of elementary rewriting steps that has quadratic order with respect to the length of the initial string.*

Proof. Let n be the length of the string to duplicate. The first step (necessarily) inserts the string between symbols S and S'. Then, the symbol S has to pass through the first n-1 symbols of the string, and this is done in linear time (n-1 steps). For any string and only one time during the process, the third rule must be applied to eliminate both S and S'. At this point, the first shadow symbol has to pass through the last $n-1$ symbols of the initial string, the second one has to pass through $n-2$ symbols, and so on, the last one does not pass through any other symbol; therefore $n-1+n-2+\ldots 0 = n(n-1)/2$ is the number of transpositions performed by the algorithm. Finally, $2n$ steps (one for each symbol of two final copies) are necessary to 'terminalize' the string.

In conclusion, $1+n-1+1+n(n-1)/2+2n$ steps are necessary (and enough) to duplicate a string of length n, therefore the complexity of TDA is $O(n^2)$. □

Without any change, this algorithm can run in parallel on many strings. Hence, parallel TDA still have a quadratic cost with respect to the length of the longest string.

We conclude the discussion about TDA with the following example of a rewriting path of the algorithm on the string *aba*.

1. aba
2. $SabaS'$ (rule 1)
3. X_aY_aSbaS' (rule 2)
4. $X_aY_aX_bY_bSaS'$ (rule 2)
5. $X_aY_aX_bY_bX_a{}'Y_a{}'$ (rule 3)
6. $X_aX_bY_aY_bX_a{}'Y_a{}'$ (rule 4)
7. $X_aX_bY_aX_a{}'Y_bY_a{}'$ (rule 5)

[4] While referring this dissertation, Gheorghe Păun suggested the following alternative set of seven (instead of eight) replacing rules. Given an input string α, let $S\alpha S'S''$ be an axiom of the system.

1. $Si \to iSi'$
2. $i'\,j \to j\,i'$
3. $i'\,S' \to S'\,i'$
4. $i'S'' \to i\,S''$
5. $S\,S' \to o$
6. $o\,i \to i\,o$
7. $o\,S'' \to \lambda$

8. $X_aX_bX_a{}'Y_aY_bY_a{}'$ (rule 5)
9. $X_aX_b{}'aY_aY_bY_a{}'$ (rule 6)
10. $X_aX_b{}'aY_aY_b{}'a$ (rule 7)
11. $X_a{}'baY_aY_b{}'a$ (rule 6)
12. $X_a{}'baY_a{}'ba$ (rule 7)
13. $abaY_a{}'ba$ (rule 8)
14. $abaaba$ (rule 9)

Duplication by dissolution (DDA)

Duplication by dissolution was studied in the context of multi-agent systems [153]. It *dissolves* a given string in a *solution*, while marking the order that any element has in the sequence. Then, any element is replicated by a specific agent, and finally the pieces floating in the solution are recombined (in two copies).

This algorithm is better expressed by rules that transform multisets of strings. Let us fix an alphabet $V = T \cup N \cup \{\bullet\}$, where T is a set of terminal symbols, N a set of numerals, and $\bullet$ a special symbol, and let α, β vary over the strings of $T^\star$, a over the symbols of T, and n, m over N.

The string of $T^\star$ that has to be duplicated is put in an initial multiset with a special symbol E, standing for an 'enzyme', that controls the process. The following rules yield two copies of the initial string (comma distinguishes the members of multisets). Each inference rule replaces a multiset having the form of the premise, with the multiset that is obtained from the conclusion, by assigning to the variables α, β, a, n, m the same values that were assigned to them in the premise.

- Enzyme binding and marking the first symbol a

$$\frac{E,\ a\,\alpha}{0\,a\,1\,1\,E\,\alpha} \tag{2.1}$$

- Marking an internal symbol a

$$\frac{\alpha\,n\,E\,a\,\beta}{\alpha\,n\,a\ \ n{+}1\ \ n{+}1\ \ E\,\beta} \tag{2.2}$$

- Releasing the string and the sequence length

$$\frac{\alpha\,n\,E}{\alpha\,\bullet,\ \ n} \tag{2.3}$$

- Splitting

$$\frac{\alpha\,n\,n\,\beta\,\bullet}{\alpha\,n\,\bullet\ ,\ n\,\beta\,\bullet} \tag{2.4}$$

- Duplicating

$$\frac{n\,a\ \ n{+}1\bullet}{n\,a\ \ n{+}1\,\bullet,\ \ n\,a\ \ n{+}1\,\bullet} \tag{2.5}$$

- Recombining

$$\frac{m\,\alpha\,n\,\bullet,\ \ n\ \ b\ \ n{+}1\ \ \bullet}{m\,\alpha\,b\ \ n{+}1\ \ \bullet} \tag{2.6}$$

- Ending

$$\frac{0\ \alpha\ n\ \bullet,\quad 0\ \beta\ n\ \bullet,\quad n}{\alpha,\ \ \beta} \tag{2.7}$$

There is no strategy to apply these rules, but if one wants to obtain strings of $T^\star$ with a process starting on strings on $T^\star$, then one necessarily performs a duplication.

Proposition 2.2. *Duplication by dissolution performs a number of elementary steps that is linear with respect to the length of the initial string.*

Proof. Let n be the length of the string to duplicate. Necessarily n steps have to be done in order that E passes through the string to mark each symbol. In one step E disappears while releasing the numeral corresponding to the length of the string. Further $n-1$ steps are performed in order to split the string into elementary pieces, that are strings such as $k\,a\,k+1$, with $k<n$. Again, a linear time (n steps) is necessary to perform the duplication of the elementary pieces by means of the fifth rule. After $2(n-1)$ steps that recombine the elementary pieces, a final step completes the duplication of the string (of length n). The complexity of DDA is thus $O(n)$. □

Advantages of such a method are the intrinsic parallelism of symbol duplication, and the fact that marking, dissolution and recombination can be performed in linear time.

The following example shows a rewriting path which starts from the string *abcd* (where $\xrightarrow{(i)}$ denotes the application of the rule i, and $\overset{(i)}{\longrightarrow}{}^*$ denotes a multiple application of the rule i).

$$\{\,a\,b\,c\,d,\quad E\,\} \xrightarrow{(2.1)} \{\,0\,a\,1\,1\,E\,b\,c\,d\,\} \xrightarrow{(2.2)} \{\,0\,a\,1\,1\,b\,2\,2\,E\,c\,d\,\} \xrightarrow{(2.2)}$$

$$\{\,0\,a\,1\,1\,b\,2\,2\,c\,3\,3\,E\,d\,\} \xrightarrow{(2.2)} \{\,0\,a\,1\,1\,b\,2\,2\,c\,3\,3\,d\,4\,4\,E\} \xrightarrow{(2.3)}$$

$$\{\,0\,a\,1\,1\,b\,2\,2\,c\,3\,3\,d\,4\,\bullet,\ 4\,\} \overset{(2.4)}{\longrightarrow}{}^* \{\,0\,a\,1\,\bullet,\ \ 1\,b\,2\,\bullet,\ \ 2\,c\,3\,\bullet,\ \ 3\,d\,4\,\bullet,\ \ 4\,\} \overset{(2.5)}{\longrightarrow}{}^*$$

$$\{\,0\,a\,1\,\bullet,\ \ 1\,b\,2\,\bullet,\ \ 2\,c\,3\,\bullet,\ \ 3\,d\,4\,\bullet,\ \ 0\,a\,1\,\bullet,\ \ 1\,b\,2\,\bullet,\ \ 2\,c\,3\,\bullet,\ \ 3\,d\,4\,\bullet,\ \ 4\,\} \overset{(2.6)}{\longrightarrow}{}^*$$

$$\{\,0\,a\,b\,2\,\bullet,\ \ 2\,c\,d\,4\,\bullet,\ \ 0\,a\,b\,c\,d\,4\ \bullet,\ \ 4\,\} \xrightarrow{(2.6)}$$

$$\{\,0\,a\,b\,c\,d\,4\,\bullet,\quad 0\,a\,b\,c\,d\,4\,\bullet,\quad 4\,\} \xrightarrow{(2.7)} \{\,a\,b\,c\,d,\quad a\,b\,c\,d\,\}.$$

Unfortunately, if one wants to duplicate many strings in parallel and in linear time by this algorithm, then one needs a 'private compartment' for each string. In fact, parallel duplication of many strings without separating them implies a quadratic cost, as established by the following proposition.

Proposition 2.3. *Parallel DDA has a quadratic cost in space.*

Proof. In order to duplicate many different strings at the same time, the marking process has to be extended in a such a way that any elementary piece is marked both with the string to which it belongs and its position inside the string. Since for each of the n symbols of a string we have to store the string itself, then we need a $O(n^2)$ space capacity of $O(n^2)$. □

Bilinear duplication algorithm (BDA)

If one uses a string as a template and put 'face-to-face' to each element a copy of it, or something that is uniquely associated to it, along a line that goes 'parallel' to the original one, then one gets an algorithm deeply different from the previous ones because it is based on a bilinear structure of strings.

Let V be the alphabet of the string α to duplicate, while $\bullet$ being a meta-symbol describing the single step of computation. Template driven algorithm or BDA can be represented by the following rules, where $\alpha, \beta, \gamma \in V^\star$, λ denotes the empty string, and the symbol x in V is such that $x \neq \lambda$.

1. Starting
$$\alpha \;\rightarrow\; \frac{\bullet\, \alpha}{\lambda}$$
2. Doubling
$$\frac{\alpha \;\bullet\; x\, \beta}{\gamma} \;\rightarrow\; \frac{\alpha\, x \;\bullet\; \beta}{\gamma\, x}$$
3. Separating
$$\frac{\alpha \,\bullet}{\beta} \;\rightarrow\; \alpha,\; \beta$$

A duplication algorithm based on a bilinear structure of strings and deterministic rewriting rules is thus obtained; in fact, in the last step we have $\beta = \alpha$. In the case of DNA duplication, in the second step a symbol uniquely associated to x (the complementary) is put 'face-to-face' to the template, and finally β results to be the Watson Crick complementary string of α.

Proposition 2.4. *Parallel BDA has a computational complexity that is linear in time and in space.*

Proof. Let n be the length of the string α to duplicate. Apart one initial and one final step, the algorithm performs n doubling steps to copy the symbols of α. Moreover, an amount of space linear with n is occupied during the whole process. □

Without any change, this algorithm can run in parallel on many strings while maintaining a linear complexity.

As an example, we consider a rewriting path of BDA starting from the string *acbdd* over the alphabet $V = \{a, b, c, d\}$ (where $\longrightarrow_*$ denotes multiple application of a rule).

$$acbdd \overset{starting}{\longrightarrow} \frac{\bullet acbdd}{\lambda} \overset{doubling}{\longrightarrow} \frac{a \bullet cbdd}{a} \overset{doubling}{\longrightarrow}_* \frac{acbdd \bullet}{acbdd} \overset{separating}{\longrightarrow} acbdd,\, acbdd$$

A different representation of bilinear duplication is the following, where an agent equipped with two arms is able to duplicate the symbols of a given alphabet and to attach the two copies at the end of its arms, as in Figure 2.4, which is taken from [76]. When all symbols of the string have been duplicated, it cuts off the two copies of the string. The agent has to know only if the last symbol which it has duplicated is the final one of the string or it does not. After the cuts each agent can be used for other strings, and by putting many agents in a solution one can duplicate many strings in parallel.

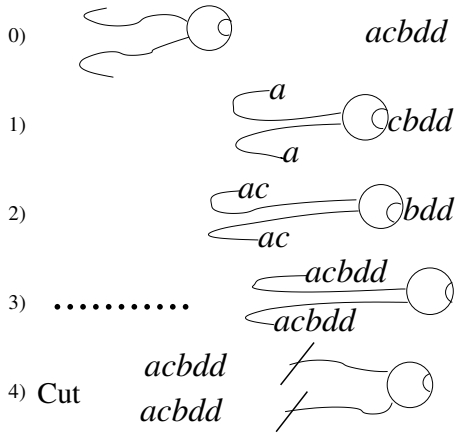


Fig. 2.4. Example of a bilinear duplication algorithm on the string acbdd.

Finally, bilinear duplication algorithm can be expressed by a function in the following way. Given a string $s \in V^\star$, let it be codified with a number. Then one transforms this number into the numerical encoding of ss, and finally one decodes it to obtain ss.

More formally, let $V = \{a_1, a_2, \ldots, a_k\}$ and $B = \{0, 1, 2, \ldots, k-1\}$. Let us define $A = \{(s, |s|) \mid s \in V^\star\}$, where $|s|$ denotes the length (number of symbols) of the string s, and $\overline{\mathbb{N}}$ as the set of natural numbers of the form $\sum_i a_i k^i$ with $a_i \in B$. Let c be any one-to-one function from V into B; for example, the following one:

$$\begin{aligned} c : V &\to B \\ a_i &\mapsto i - 1, \end{aligned}$$

and i, g, h the following functions:

$$\begin{aligned} i : \quad & V^\star \to A \\ & s \mapsto (s, |s|) \end{aligned}$$

$$\begin{aligned} g : \quad & A \to \overline{\mathbb{N}} \\ & (s_0 s_1 \cdots s_{n-1}, n) \mapsto \textstyle\sum_{i=0}^{n-1} c(s_i) k^i \end{aligned}$$

$$\begin{aligned} h : \quad & \overline{\mathbb{N}} \to \overline{\mathbb{N}} \\ & \textstyle\sum_{i=0}^{n-1} a_i k^i \mapsto \sum_{i=0}^{n-1} a_i (k^i + k^{n+i}) = \sum_{i=0}^{2n-1} a_i k^i, \quad a_i = a_{n+i} \ \forall i \end{aligned}$$

Because both i and g are one-to-one functions, we can define the duplication function

$$f = i^{-1} \circ g^{-1} \circ h \circ g \circ i$$

such that $f(s) = ss$ for all $s \in V^\star$.

This function could be implemented by an agent equipped with two arms capable to keep a mutual distance, as in Figure 2.5, which is taken from [76]. It computes the length n of a given string, then makes a structure with $2n$ *loci* having a separation between the $n-th$ and the $(n+1)-th$ locus, and puts the initial string in the first part of the structure. Finally, the agent copies each symbol in the second part by moving forward step by step the left (reading) arm and the right (writing) arm (at mutual a distance n).

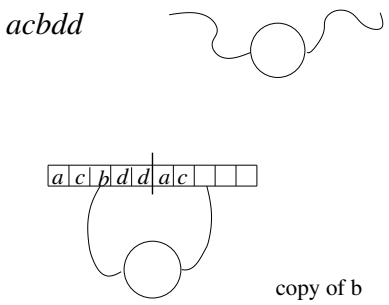


Fig. 2.5. Example of a bilinear duplication algorithm on the string acbdd.

2.3 DNA pairing

Nature employs the bilinear duplication algorithm to copy DNA strings with a computational complexity linear in time and space. In this section, we show that if one wants to implement the template driven duplication algorithm by means of double linear arrangements of molecules that are floating in a fluid support, as it is the case of DNA strings, then one needs molecules that are asymmetric in the three space directions, and arranged bilinearly according to the antiparallelism and complementarity principles. In other words, duplication of mobile sequential structures, requires double strings organized according to the constitutive principles of DNA.

First we reason on the idea of floating string and floating double string. Formal string is the basic notion in the digital elaboration of information. From a mathematical point of view, a string α is a function from a segment of natural numbers $\{1, 2, \ldots, n\}$ to a finite set of symbols A called alphabet. The number n is the length of the string (the empty string λ is assumed to have length 0). In an explicit way, α is also represented (according to the usual west standard order from left to right) by writing in the first position from the left the value $\alpha(1)$ that α associates to 1, then on the right of it, the value $\alpha(2)$ that α associates to 2, and so on, with the value $\alpha(n)$ put in the rightmost position:

$$\alpha(1)\alpha(2)\alpha(3)\ldots\alpha(n).$$

But, this notation hides a crucial point[5]. We can define α in this way because we assume that in writing and in reading the symbols on the paper they remain in the position they were placed on, and also that the page has a standard orientation with respect to the eyes of the writer and of the reader. If one is not allowed to make these assumptions, this notation is useless. Even if all the symbols remain together, while leaving unchanged their relative position, one cannot uniquely establish what is the sequence associated to the linear arrangement given in the notation above. In the case of mobile symbols indeed, the only way to recover the linear order implicit in the notation above is to have an orientation, say left-right, in the symbols, in such a way that given a linear arrangement, we can distinguish the first symbol of this structure, say the symbol having nothing on the left, from the last symbol, having nothing on the right.

Concatenation (usually indicated by juxtaposition) can be applied only between strings oriented in the same way (from a biochemical viewpoint, ligase enzyme performs the 'fusion' only of $\rightarrow$ or $\leftarrow$ links, while working as in Figure 3.7, which is taken from [85]). Indeed, we can assume an axiom regarding the concatenation direction:

- **Concatenation Direction.** Mobile strings require that each molecule has to be asymmetric with respect to a 'concatenation direction' and concatenation has to be performed between strings with the same verse along this direction.

The usual reverse operation on strings $rev(a_1\ldots a_{n-1}a_n) = a_n a_{n-1}\ldots a_1$ in fact reverses the concatenation verse of a string.

In order to pair *oriented strings*, mobile symbols that are already oriented along a direction, say x, need to possess another direction, say y, where two pairing symbols connect. If we assume that no more than two floating strings can be paired, then each symbol must have only one pairing link in the direction y and this implies an asymmetry with respect to this direction.

- **Pairing Direction.** Bilinearly paired molecule strings require that each molecule has to be asymmetric with respect to a pairing direction.

In fact, assume to have two pairing links in the y direction (symbols shaped in the following way $\updownarrow a$), in this case we could obtain multiple paired structures of the following type:

$$\frac{\begin{array}{c}\updownarrow\alpha\\ \updownarrow\alpha'\end{array}}{\begin{array}{c}\updownarrow\alpha\\ \updownarrow\alpha'\end{array}}.$$

Instead, if one has the strings $\alpha = abaa$ and $\alpha' = a'b'a'a'$ in the double structure $\frac{\alpha}{\alpha'}$, then it must have the following shape:

$$\frac{\downarrow a \downarrow b \downarrow a \downarrow a}{\uparrow a' \uparrow b' \uparrow a' \uparrow a'}$$

[5] J. Bennett. The difference between right and left. *American Philosophical Quaterly*, 7:175-191, 1970.

or in a more compact way:

$$\frac{\downarrow \alpha}{\uparrow \alpha'}$$

and the verse along the y direction is essential in order to have a binary pairing.

The following alphabets comprise of the symbols from an alphabet V where concatenation and pairing verses are explicitly indicated.

- $\uparrow V \rightarrow \stackrel{def}{=} \{\uparrow w \rightarrow \mid \ w \in V\}$
- $\downarrow V \rightarrow \stackrel{def}{=} \{\downarrow w \rightarrow \mid \ w \in V\}$
- $\uparrow V \leftarrow \stackrel{def}{=} \{\uparrow w \leftarrow \mid \ w \in V\}$
- $\downarrow V \leftarrow \stackrel{def}{=} \{\downarrow w \leftarrow \mid \ w \in V\}$.

Therefore, we can define the *floating* or *mobile* strings as the elements of the following language [76]

$$W^{\diamond} = (\uparrow V \rightarrow)^{\star} \cup (\downarrow V \rightarrow)^{\star} \cup (\uparrow V \leftarrow)^{\star} \cup (\downarrow V \leftarrow)^{\star}.$$

Actually, for notational convenience, in this thesis we will not use this terminology, but we continue to use the standard implicitly oriented letters to denote symbols of alphabets. However, these speculations help to be aware of what there is behind our usual formalizations, and highlight asymmetries of the molecule that are not just due to chemical reasons but also to informational reasons.

Now we consider briefly the algorithm BDA in the context of the mobile strings, while further and more formal argumentations can be found in [76, 148]. We know from the physical implementation of chemical bonds that those acting between two equal molecules are covalent bonds, that are very strong. In fact, for reasons of symmetry, the common electrons responsible of the bonds are uniformly shared by the two components. But, as we have seen in the previous section, the pairing duplication is based on the possibility of separating the two components of a double structure, in order to 'recover' from each of the strings the other one.

This means that the basic step of BDA duplication $\frac{\alpha \bullet w\, \beta}{\gamma} \ \rightarrow \ \frac{\alpha\, w \bullet \beta}{\gamma\ w}$ results an oversimplification, and has to be replaced by $\frac{\alpha \bullet w\, \beta}{\gamma} \ \rightarrow \ \frac{\alpha\, w \bullet \beta}{\gamma\, h(w)}$ where h is a map such that $h(x) \neq x$ for any symbol x, and it is an involution (h^2 is equal to the identity) that associates mobile symbols to mobile symbols with opposite y direction. In fact, when the algorithm starts on a string α and, after the separating step, it obtains the couple $(\alpha, h(\alpha))$, it continues to work on both strings, so obtaining a first couple $(\alpha, h(\alpha))$ from α, and a second couple $(h(\alpha), h^2(\alpha))$ from $h(\alpha)$ (and so on). It is clear that the algorithm performs a duplication only if $h^k = I$ for some $k > 0$, and in this case k steps are necessary for obtaining a second copy of the initial string. Therefore, in order to satisfy the condition $h(x) \neq x$ the minimum value of k is 2, and the function results to be bijective.

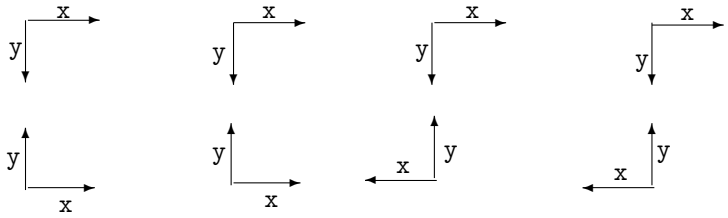
The function h must to be extended to a morphism on the floating strings, we still call it h, in order to express the duplication process of the bilinear duplication algorithm, where the lower string is obtained by applying h to each symbol of the upper string and then by concatenating the results. More precisely, $h(\alpha\beta) = h(\alpha)h(\beta)$ for all α, β strings of $W^{\diamond}$.

As a consequence of the existence of such a function h, the alphabet of mobile symbols (on which it is an involution and it is different from the identity for every argument) must obviously have an even number of elements. Two would be the minimal number for satisfying all the principles on which pairing duplication is based on, but nature uses four elements. However, since a double linear structure is composed of two single complementary strings, if these strings vary over an alphabet with just two symbols, then all the double linear structures with the same length would be equivalent. In other words, if we use four symbols, then we get the advantage to have an additional binary code written on the double stranded strings (if 0 expresses the pair bond A-T and 1 the pair bond C-G, then a double stranded DNA molecule identifies a binary string) [76]. It seems reasonable that nature does not underestimate this informational advantage. Namely, conformational aspects seem linked to this second code.

Antiparallelism

We notice that floating symbols need to be asymmetric with respect to a third (with respect to concatenation and pairing) direction, in order that each strand of the double structure is the h-image of the other one.

In fact, if mobile symbols were two-dimensional objects or symmetric with respect to the third space direction, say z, then after turning up-down the verse of the z axis, we would get the same shape. In this case only the two directions x and y would be relevant for catenation and pairing (one could avoid to consider the z direction). This possibility though would allow us to get such kinds of pairings:



where, the upper string is paired with two lower strings that cannot be concatenated because they do not have the same catenation verse. Since any lower string is the image of the corresponding upper string, under the morphism h from $W^\diamond$ to $W^\diamond$, a string can be paired only along a line 'parallel' to the 'catenation line' and with only a string placed along this line.

Therefore, we can assume to orient the z axis of a molecule in such a way that the pairing site of the molecule is placed in the region of points with the z coordinate strictly positive.

- **Third direction.** Symbols of mobile strings must have a third direction, say z, and must be asymmetric with to respect to this direction.

Moreover, for biochemical motivations (for example, enzymes read both strands of molecules from the same side) it is reasonable to suppose that the pairing connection is established between sites that are located at a 'minimal distance'.

That is, we assume that floating symbols while pairing have the same verse of the z direction.

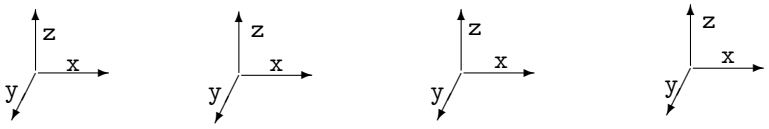
Definition 2.5. *A molecule representing a symbol is chiral (from a Greek etymology for 'hand') when it is asymmetric with respect to the three space directions, that is, its structure can be used for defining a space coordinate system.*

Our mobile symbols we call 'molecules' are chiral, but we can distinguish two possible situations:

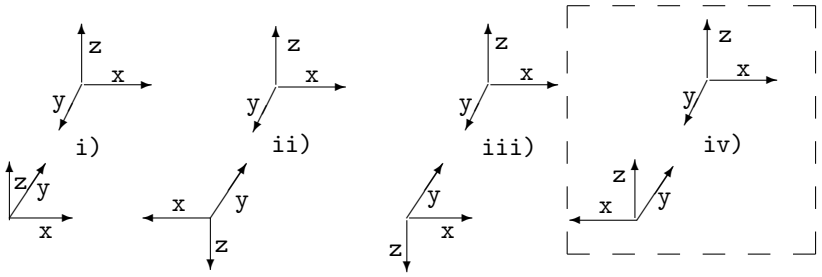
- *homogeneous chirality* when coordinate systems of our molecules are all right-handed or all left-handed.
- *heterogeneous chirality* when both types of right-handed and left-handed molecules are present.

Proposition 2.6 (Antiparallelism). *The hypothesis of homogeneous chirality implies antiparallelism.*

Proof. Without loss of generality we can assume that our molecules are all right-handed. Consider the following string (where the individuality of the single molecules is not indicated):



then, consider the ways we can pair the molecules of this string with other molecules. In general we have four possibilities expressed in the following picture.



It is easy to check that in the pairings i) and ii) the molecules on the lower line have chirality different from the molecules of the upper line. Therefore, under the hypothesis of homogeneous chirality, these pairings are impossible. Moreover, the pairing iii) violates the assumption of a common z-verse for pairing molecules, because the pairing site of the lower molecule is not located in the z positive region with respect to the coordinate system of the upper molecule.

Therefore, the pairing iv), put inside the frame, is the only way to pair two strings of molecules, oriented and well concatenated, in such a way that their pairing sites are at the minimal distance. But this pairing is just performed according

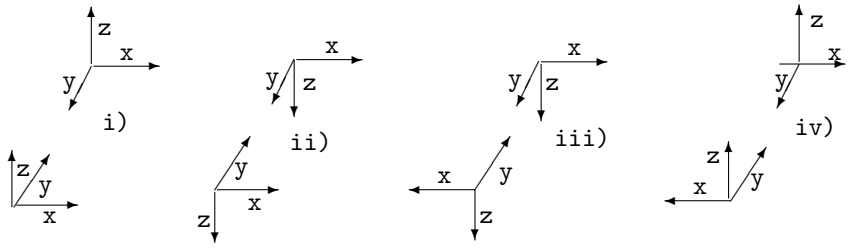
to the well-known antiparallel concatenation verse. □

In a very intuitive way, the situation described above is analogous to what happens in a folk dance where a line of dancers is paired to another line of dancers. The concatenating verse of x is from the left to the right of the body, where hands join, the pairing verse of y is from the back to the face of the body, and the z direction is the vertical axis with the usual verse. The pairing sites are on eyes, so that each dancer pairs with the opposite dancer by meeting his/her sight. Each line has a catenation start where there is a dancer with nobody on the left side. The *chirality* of human body implies that there is no arrangement different from the antiparallel way for matching the two lines of dancers, in the sound (face-to-face) correspondence [76].

The case of homogeneous chirality corresponds to the real case of DNA, while in the case of heterogeneous chirality the antiparallelism is replaced by a more complex principle:

Proposition 2.7. *Under the hypothesis of heterogeneous chirality, a molecule can pair either only with a molecule with different chirality in a parallel way, or only with a molecule of the same chirality in an antiparallel way.*

In fact, if both right-handed and left-handed molecules can belong to the string in the (for example) upper line of a double structure, then, from all the above assumptions, the possible pairings are reduced to ones in the following picture.



Therefore, a molecule can only pair either with a molecule with different chirality in a parallel way, cases i) and ii), or with a molecule of the same chirality in an antiparallel way, cases iii) and iv). If both possibilities are allowed, then we would have molecules with different catenation verses in the lower line of a double string structure, that is forbidden by the correctly working of the bilinear duplication algorithm.

In this dissertation, as usual, the length of a string α is denoted by $|\alpha|$, and its pairing (complementary and with opposite concatenation orientation) string by $\overline{\alpha}$.

2.4 DNA-like computations

The rules of the bilinear duplication algorithm that are described in Section 11 have a clear biotechnological implementation. In fact, the doubling step can be performed by a polymerase enzyme, and the separating step by a denaturation

operation, due to an increase of the temperature. While the first step is just the formal passage from a single string to a string placed into a double structure. The biological interpretation of the symbol • is related to the orientation of the string: the computation starts only when it is at the beginning of a string with respect to the concatenation verse, enzymes can recognize this point from the chemical structure of the molecule.

In this section we propose a universal (DNA-like) system which computes on double structures of mobile strings by means of rewriting rules that have a biotechnological implementation. Essentially three types of operations are enough in order to obtain universal computations, as one can see from [160,161,162]; they are duplication, triplication, and division by six. Our system processes the information encoded on floating strings by simulating these three operations. It is able to execute any program of a register machine, since it has also a mechanism to switch from one instruction to another one and to perform jump instructions.

This system is an extension of the floating bilinear duplication algorithm. In fact, (from a theoretical point of view) the computation can be implemented on DNA strings; but the crucial difference is the encoding used to represent numerical information by a floating string. The output of the bilinear duplication is given by two copies of the initial string, and from the same viewpoint, the product of the triplication would be given by three copies of the initial string and the division by six would reduce six copies of the string to one copy. That is, the numerical information which is processed by the operations consists of the number of copies of the initial string. Here instead, we consider the information as completely carried by (one copy of) the current string, regardless to the number of copies of such a string. It is transformed by means of operations in each step of computation, as usually on the tape of a Turing machine. The initial string encodes the given numerical input, and when one of the operations modifies it, the string encoding the number obtained after the application of that operation is produced.

Let V be the alphabet with four symbols $\{1, 2, 3, \#\}$, and n a numerical input. The factorization in primes of n can be seen as $2^k m\, 3^h$, with $m \in \mathbb{N}$ and non-divisible by 2 or 3. Since every m can be written in unary code, then every numerical input n can be considered as a particular string over V, with a number k of 2s as prefix and a number h of 3s as suffix that are uniquely implied by the value of n. In particular, we associate n to the string $\downarrow 2^k \# m \# 3^h \rightarrow$, and we refer to it simply with $2^k \# m \# 3^h$ belonging to $V^\star$.

A DNA-like system

We present a formal system based on the double string structure of mobile strings, that is able to duplicate, triplicate, and divide by six a number n (represented by a string encoding as above), and to switch from performing one instruction to another one of a register machine program. By [159,160,161,162] we know that this suffices in order to perform universal computations, as all the basic instructions of a register machine can be reduced to "times 2" or "times 3" or "division by 6" instructions.

With our encoding, in order to duplicate a number it is enough to juxtapose the symbol 2 at the beginning (with respect to the concatenation verse) of the

string encoding the number, and in order to triplicate one has to concatenate the symbol 3 at the end. The division by six is performed by eliminating the first and the last symbol of the string if both of them are different from #, otherwise the number is not divisible by six.

The system works with deterministic rewriting rules over double string structures, where a special meta-symbol of kind $\bullet$ marks the point of progress of the computation. In particular, we consider different pointers corresponding to the p instructions of the program we want to simulate, they belong to the set Q of pointers and have an index representing an instruction label, $Q = \{\bullet_1, \bullet_2, \ldots, \bullet_p, \ldots\}$. In the case of instructions which are decrements, that correspond to divisions [159], during the computation we use further (primed) indexes, among $\{1', \ldots, p'\}$; however the number of our pointers, that is the cardinality of Q, is bounded by $2p$, where p is the number of the instructions of the given program.

In the following, we have three groups of rewriting rules, where λ denotes the empty string (in $V^\star$), $\alpha, \beta, \gamma, \xi \in V^\star$ with $\beta, \xi \neq \lambda$, whereas $a, b \in V$, $j \in \{1, \ldots, p\}$ and $j' \in \{1', \ldots, p'\}$. We refer to ξ' to denote the string obtained from ξ after having erased its first symbol.

1. Starting

$$2^k\#m\#3^h \longrightarrow \frac{\bullet_1\, 2^k\#m\#3^h}{\lambda}$$

This step marks the start of the computation, by transforming the input from single string into double string structure. After such a step the first instruction is going to be executed, as required by the presence of the pointer with index 1.

2. This multiple step performs the operation indicated by the jth instruction of the program (whenever it can be performed), for $j \in \{1, \ldots, p\}$, that can be a duplication, a triplication, a jump, or a division by six.

   a) Doubling (for j belonging to the labels such that the corresponding instructions are duplications)

      i.
      $$\frac{\bullet_j\, a\, \alpha}{\lambda} \longrightarrow \frac{a\ \bullet_j\ \alpha}{2a}$$

      ii.
      $$\frac{\gamma\ \bullet_j\ a\, \alpha}{2\gamma} \longrightarrow \frac{\gamma\, a\ \bullet_j\ \alpha}{2\,\gamma\, a}$$

   b) Tripling (for j belonging to the labels such that the corresponding instructions are triplications)

      i.
      $$\frac{\alpha\ \bullet_j\ a\, \beta}{\alpha} \longrightarrow \frac{\alpha\, a\ \bullet_j\ \beta}{\alpha\, a}$$

      ii.
      $$\frac{\alpha\ \bullet_j\ a}{\alpha} \longrightarrow \frac{\alpha\, a\, \bullet_j}{\alpha\, a\, 3}$$

   c) Jumping from j to l (if the label j corresponds to an instruction which is a jump to the l-th instruction)

$$\frac{\bullet_j\, \alpha}{\lambda} \longrightarrow \frac{\bullet_l\, \alpha}{\lambda}$$

d) Dividing by six if possible, otherwise *jump from j to l* instruction (j belonging to the labels such that the corresponding instructions are divisions by six, which can be performed if, and only if, both the first and the last symbol of the current string are different from #)

i.
$$\frac{\bullet_j\, 2\, \alpha}{\lambda} \longrightarrow \frac{2\ \bullet_j\ \alpha}{\lambda}$$

ii.
$$\frac{\bullet_j\, \#\, \alpha}{\lambda} \longrightarrow \frac{\bullet_l\, \#\, \alpha}{\lambda}$$

iii.
$$\frac{\xi\ \bullet_j\ a\, \beta}{\xi'} \longrightarrow \frac{\xi\, a\ \bullet_j\ \beta}{\xi'\, a}$$

iv.
$$\frac{\xi\ \bullet_j\ 3}{\xi'} \longrightarrow \frac{\xi\, 3\, \bullet_j}{\xi'}$$

v.
$$\frac{\xi\ \bullet_j\ \#}{\xi'} \longrightarrow \frac{\xi\, \#\, \bullet_{j'}}{\xi'\, \#}$$

vi.
$$\frac{\xi\, \bullet_{j'}}{\xi'} \longrightarrow \frac{\xi\, \bullet_{j'}}{\bullet_l\, 2\, \xi'}$$

3. Switching and Separating

a)
$$\frac{\alpha\, \bullet_j}{\beta} \longrightarrow \frac{\alpha\, \bullet_j}{\bullet_{j+1}\beta}$$

b)
$$\frac{\alpha\, \bullet_i}{\bullet_j\, \gamma} \longrightarrow \frac{\bullet_j\, \gamma}{\lambda}$$

c)
$$\frac{\bullet_{p+1}\, \gamma}{\lambda} \longrightarrow \gamma$$

Whenever a string which is preceded by the pointer $\bullet_{p+1}$ is transformed in a single string (that is the step 3 part (c) is executed), then the program halts and the single string encodes the numerical result of the computation.

Given a register machine program, all its basic instructions of increasing by 1, decreasing by 1, and testing if a register contains 0 or not, may be simulated by a program with p labeled instructions that double, triplicate, or divide by six the numerical information, or represent a jump. In the above system the first step is performed only at the beginning of the program. Then, whenever the current label j points to a doubling or a triplicating instruction, the rewriting rules of the step 2,

part (a) and part (b) respectively, are applied in a deterministic way. In both these cases the operation is completed when no other rule related to that operation can be applied, and a double structure of kind $\frac{\alpha\bullet}{\beta}$ is finally obtained. If label j refers to a jump, the rule of the step 2 part (c) is applied, so that the operation labeled by l can be performed. If j corresponds to a division by six, then the rewriting rules of the step 2 part (d) are applied. If the current string begins with the symbol 2 and ends with the symbol 3 (that means that it represents a number divisible by six, and the division can be performed), then the computational process leads to obtain a double structure of kind $\frac{\alpha\bullet}{\beta}$. Otherwise, a jump is performed whenever the first symbol is equal to #, and a double structure of kind $\frac{\alpha\bullet}{\bullet\beta}$ is obtained whenever the last symbol is equal to #.

The switching and separating rules of step 3 lead deterministically the computation to perform the next operation, or to give out an output. In particular, the (a) rule switches to the next instruction (the $(j+1)$th one), after that the operation j has been performed regularly, while the (b) rule allows the corresponding instruction to start even after an operation of division by 6 which was not performed. The (c) rule is applied at the end of the program, and it gives out the current string as output.

Note that we have an alphabet with exactly four symbols and the typical DNA string double structure. Moreover, the biochemical implementation of the main step (the second one) recalls the work of a special polymerase, since symbols are copied from a template by elongating a primer with concatenation, and the very last step recalls the annealing process.

In the case of duplication, a translation of the first symbol a into the string $2a$ has to be done (i.e., a symbol 2 and a symbol a are juxtaposed as lower string of the double structure) before performing the copying of the remaining string, and in the case of triplication, a translation of the last symbol a into the string $a3$ has to be done after a usual copying of the string. While in the case of division, the copying is performed after the elimination of both the first and the last symbol of the string.

There are some enzymes called exonuclease (see Section 5.2) with the ability to cut away the extremal symbols of a string. Unfortunately, from an experimental point of view, it is really difficult to control these enzymes in such a way that only one letter is removed. At a theoretical level though, in all these cases, the natural orientation of the DNA strings is a mechanism to recognize which is the first symbol and which is the last one of a string, and enzymes as polymerase or exonuclease are able to recognize these points.

The exchange of pointers and the appearing of a pointer in the lower string can be seen as indications corresponding respectively to what enzyme is going to read and copy and from where. In DNA computations, the enzymes are chosen by humans, and the point of progress of the computations is given by the natural orientation of DNA strings.

2.5 Membrane systems

To achieve the realization of microscopic computers, it is necessary to learn as much as possible from the biology of the living cell, that is currently the only known DNA-based molecular computer in existence. Cells were observed for the first time about three hundreds years ago, right after the construction of the first microscopes. In the second half of the nineteenth century it was already clear that they represent the smallest and irreducible unities of life of which every organism is comprised. Eukaryotic cells keep together the genes, the RNAs with the proteins which they encode, as well as the products of their activities. Cellular compartmentalization is the key of the organization of biomolecular systems, such as networks of proteins underlying the main cellular functions, and it is essential for the evolution of all living organisms.

Models based on abstract process languages use the 'compartment-as-process' abstraction. Though the most inherent works, such as Petri Nets [92] or process algebra [172], do not handle compartments explicitly, with some exceptions, where abstraction of movement of and between compartments is however either limited or completely absent [165, 192].

In this section membrane systems are proposed as a model to represent various aspects of molecular localization and compartmentalization, including the movement of molecules between compartments, the dynamic rearrangement of molecular reactions, and the interaction between molecules in a compartmentalized setting (for a brief introduction to the model see Section 1.1.1, and for a picture of a membrane structure see Figure 2.6).

There exist similar models motivated by ambient calculus, a process algebra for the specification of process location and movement through computational domains, that have a flavor of biology [192, 34]. Indeed, in this paradigm of mobile computation [35], each ambient harbors a collection of processes, that reside and run directly within it, and the processes inside an ambient control it, by instructing it to move. The biochemistry of the cell works differently though. The nature of each process (which can be a transformation, a duplication or an expelling of molecules across cell membranes) is relevant as it changes the environmental conditions for the other processes and not only in the same compartment. For example, until DNA has been 'translated' into mRNA the corresponding protein cannot be synthesized, because the information is not present yet. Moreover, the organization of compartments do not depend directly (often at all) on the processes inside them (e.g., the nucleus of a cell is there in any case). Cell membranes can duplicates, can be dissolved and created, and usually it depends on environmental conditions. Finally, an important point of biological relevance is the interest to the molecular reactions themselves rather than only to their distribution and coordination in compartments. Therefore, in this context, considering computational processes as elementary objects would be quite reductive.

Membrane systems, also named P systems, were inspired by cell biology in their formulation[6]. Since they were introduced [178], they have been widely investigated as models and tools of interest for computer science, and although still "dynamic

[6] "Membrane computing is a branch of natural computing which abstracts from the structure and the functioning of living cells", Gh. Păun, first sentence of the book [180].

P systems are not well developed, rendering them limited in their applicability to biological systems" (A. Regev, E. M. Panina, W. Silverman, L. Cardelli, and E. Shapiro, [192]), many works have successfully advanced the state of the art in this direction [43], and the intent to employ membrane systems as a framework to study real biological systems is vividly pursued [24, 149, 12, 18, 66, 25] ("Making use of both the theoretical developments and of the existing implementations, MC started to be used as a framework for modelling biological processes, especially at the level of the cell." Gh. Păun, [181]).

A *membrane structure* is a string of matching enumerated brackets; such as:

$$[_1\ [_2\ ]_2\ [_3\ ]_3\ [_4\ [_5\ ]_5\ [_6\ [_7\ ]_7\ ]_6\ ]_4\ ]_1$$

that can be represented as well by the diagram in Figure 2.6.

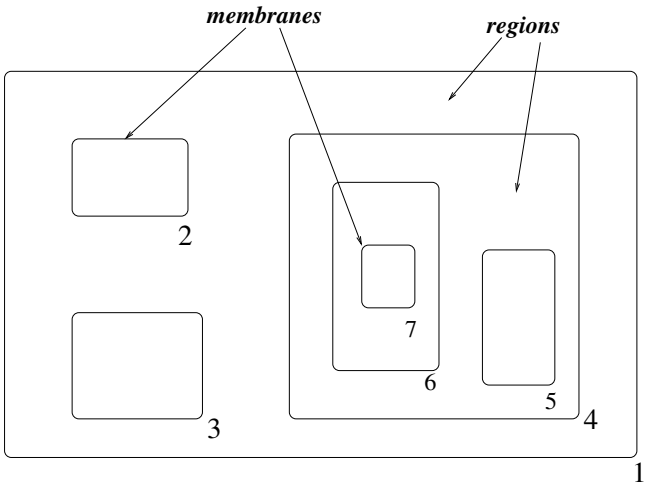


Fig. 2.6. Example of a membrane structure, where membranes and regions are pointed out.

An *evolution rule* is a string rewriting rule of the form

$$u \rightarrow v$$

where u is a string over an alphabet O, and v is a string over $O \times TAR$, where $TAR = \{here, out\} \cup \{in_j | 1 \leq j \leq m\}$. These rules represent the chemical reactions residing in the compartments of a cell. For example, the rule $abc \rightarrow (b, here)(d, out)(c, in_k)$ means that three elements a, b and c react, producing a copy of b, d, and c. Moreover, b remains in the same region, d is sent outside of the compartment, into the surrounding region, and c enters the directly inner membrane labeled with k [181].

A standard definition of membrane system is the following, from [180].

Definition 2.8. *A membrane system of degree m, where $m \geq 1$, is a structure*

$$\Pi = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_o)$$

where O is an alphabet, and its elements are called objects, μ is a membrane structure, with the membranes (and hence the regions) labeled by $1, 2, \ldots, m$. For $1 \leq i \leq m$, w_i and R_i are respectively the multiset of objects and the finite set of evolution rules over O associated to the region i of μ. Finally, i_o belongs to $\{1, 2, \ldots, m\}$ and denotes the label of an elementary membrane, called output membrane.

Membrane systems are a discrete device (as only discrete quantities are involved) which takes explicitly in account the notion of compartments. The way how the evolution rules work recalls the computational strategies of formal grammars, that traditionally were applied to the analysis of biological structures [142].

Most recently, membrane systems were differently defined in [34], as a notion grounded in process calculi that assumes concepts of metric and regularity of functions. "A membrane is a curve in $\mathbb{R}^2$ that is closed, non-self-intersecting, and smooth. A membrane system is a collection of membranes such that no two membranes intersect", L. Cardelli, [34]. In this context, the Jordan's Curve Theorem is called to assert that any membrane divides the plane into a bounded connected *inside* region and an unbounded connected *outside* region.

If one thinks of chemical reactions that involve the same molecules and happen in parallel inside cytoplasm and nucleus of a cell, topological features of a model seem to be important to simulate these processes. The concept of distance though is not necessary, at least at a certain level of abstraction. Let us explain by an example: we suppose to have two molecules that interact (namely by transforming each other) just by contact, instead of sending signals to each other. In this case, having a distance would be important; in fact, we could evaluate their relative distance in any computational step of the system, and assume that they interact when this distance is zero. Such an approach allows even to estimate in how many steps the two molecules meet each other in the system. On the other hand, in this way we enter very much in details of processes, by loosing the ability to observe the global properties of the system. It might be better to say that those specific molecules interact *eventually*, given that they stay in a same compartment (i.e., 'close enough'), no matter when and how exactly. These speculations recall the concepts of microscopic indetermination explained in Section 1.1 and the basic strategy of forbidding-enforcing systems (see Sections 1.1.1 and 7.1).

Usually biomolecular phenomena are continuous in space and time. Though, in most cases, the information necessary to describe them is inherently discrete (think, for instance, of DNA replication). There are also cases in which a discrete characterization of this information generates errors, but such errors can be arbitrarily reduced or, equivalently, the precision is proportional to the granularity with which the continuous phenomenon is reproduced by the discrete model.

2.5.1 Some variants

The original model of membrane system was enriched with several other features, usually inspired by biology. We present here very few examples from the numerous variants that may be found in literature.

A P system is called *cooperative* if it contains evolution rules with radius greater than one, where the radius of a rule $u \to v$ is the length of u. A particular class of

cooperative systems is that of *catalytic* systems, where the rules with radius greater than one are activated (i.e., can be applied) by the presence of special '*catalyst* objects', which are not transformed. This kind of rules formalizes what happens in the cell when some enzymes catalyze a molecular process that is enzyme enhanced. In the cell there are also chemicals that, while supporting or forbidding certain reactions, can separately evolve, in parallel with the chemicals involved in the reactions. These features were formalized in the context of membrane systems, by means of *promoters* and *inhibitors*, that are respectively permitting and forbidding objects associated to regions [27] (*activators* are a recent variant of catalysts and promoters). Other powerful extensions of the membrane formalism are represented by the introduction of a *priority* relation among the rules from each region, in the form of a partial order relation, and by a *polarization* (associated to membranes) that controls the activation of some rules with respect to other ones.

The *symport/antiport* rules introduced in [176] are related to the exchange of objects between membranes: each membrane can communicate with other membranes by sending out objects that the other membrane can bring inside. A symport rule has the form (x, in) or (x, out), and an antiport rule has the form $(x, in; y, out)$. These rules have a *weight*, which is given by $|x|$ in the fist case, and by $\max\{|x|, |y|\}$ in the second case. Unlike most of the other variants of P systems, this family has the feature that no objects of a system change during the computation; they only change their locations related to membrane label numbers in the system. For different types of P systems with symport/antiport rules it was found that there is not difference between the computing power of deterministic and nondeterministic systems. The symport/antiport systems were then extended by evolution-communication P systems, where the objects are also transformed by means of evolution rules.

The symport/antiport rules were generalized by the *boundary rules* introduced in [19], that are not 'located' into regions but properly on the border of the membrane. In fact, the boundary rules are not 'blind': they are not internal to regions but they are able to 'see' even externally, that is, they are sensible to what happens around the corresponding membrane. In a sense, membranes with boundary rules have 'windows' to look at the environment. An analogous idea was developed by [78] in another context, in terms of P systems with rules 'assigned to membranes'. In such a work, the strings involved in the splicing operation are either taken from inside or from outside the membrane, and the strings resulting from the splicing operation also are able to go inside or outside the membrane. Together with splicing operations, also operations of cutting and recombination were used as rules assigned to membranes [78].

Finally, we recall the 'conformon P systems' as a simple variant of P systems; they replace the objects with ordered pairs of name (object) and (numerical) value, that is called conformon. They are universal systems introduced in [82] and recently implemented in [83]. In fact, in most cases, membrane systems exhibit Turing universality and fast solutions to hard problems. The definition of complexity classes of P systems is huge matter of investigation, whereas the descriptional complexity measurements are usually given by the number of membranes, the size

of rules, the number of objects. P systems with *active membranes*[7] and only two polarizations are universal and even able to solve SAT in linear time. Finally, a universality result has been obtained for systems with only two membranes, using only multiset evolution rules and sending objects out of a membrane while changing its polarization [3].

PB systems were proposed as a variant of P systems with boundary rules in [19]. The authors proved the computational universality for PB systems with three membranes which regard at most one symbol outside and one symbol inside a membrane. They also investigated the relationships with the basic model of P systems by proving an equivalence between P systems without priority, thickness or dissolution operator[8], and PB systems that use communication rules of restricted form. The boundary notation $\alpha\ [_i\ \beta\ \rightarrow\ \alpha'\ [_i\ \beta'$ means that if the i-membrane contains the multiset β and the multiset α is present outside, then the multisets α' and β' are produced outside and inside the i membrane, respectively. The boundary notation can be extended naturally in several directions to cope with typical biological phenomena. Namely, the idea of membrane transport is well simulated by explicitly introducing membranes in the rules, as "boundary entities" which control inside and outside of the delimited region [19]. In PB systems, communication is allowed by general rules of the form $\alpha\ [_i\ \beta \rightarrow \alpha'\ [_i\ \beta'$, because if the membrane i contains the multiset β and the multiset α is present outside the membrane i, then a "communication" can be established by producing the multisets α', β' outside and inside the membrane i, respectively. Moreover, with respect to [176], transformation rules $[_i\,\beta \rightarrow [_i\,\beta'$ are allowed in PB systems.

An advantage of boundary notation is a more compact definition of P systems. In fact, according to [180] and to the definition given in the previous section, a P system is a structure $\Pi = (A, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_o)$ that is, alphabet, membrane structure, words inside regions, rules relative to regions, and output membrane. If we use the boundary notation, we can define a P system in a more concise way[9] as

$$\Pi = (O, \mu, R, i_o)$$

with O alphabet, μ the whole initial membrane system configuration (including both membranes and multisets of objects), R a finite set of boundary rules that (apart the objects which they transform) contain explicitly in their expression also on which membrane of the system they work, and i_o the output membrane. If we are studying the life of the system instead of what language it computes (that is, we are interested only to the evolution of the system from the initial configuration μ), then we can even consider just the structure $\Pi = (O, \mu, R)$.

In [75] we extended the boundary notation to describe new kinds of rule suggested by typical phenomena occurring in cells. A first task has been the formalization of the very important concept of cellular receptors, which are pro-

[7] They are membranes able to duplicate, or, more in general, membranes with the ability to dynamically change the membrane structure μ of the system [178].

[8] 'Thickness' and consequent permeability to entrance of objects can be associated to membranes by means of operators [180].

[9] As suggested in: G. Franco, V. Manca, *String Models and String Dynamics*, TR of the EMCC (European Molecular Computing Consortium) Meeting, p 17, Turku (Finland), May 15-17, 2003.

tein three-dimensional forms that recognize and fit with their specific counter-receptors. For example, the immunological process of the selective recruitment of leukocytes starts when the receptors located on leukocyte cells surface bind with some 'counter-receptors' located on the surface of endothelial cells, and these bonds slow down the initial speed of leukocytes in the blood.

From a membrane computing point of view, the notion of location due to the presence of compartments needed to be extended. In fact, objects in the membrane system usually may reside either inside or outside a given compartment. Receptors instead reside across a membrane boundary, and in a sense belong to two compartments, because they can 'see' both internal and external regions. In [75] this particular phenomenon was studied and a notation in double bracketing fashion $[_i\,[_i \quad]_i\,]_i$ allowed us to deal with 'receptors' (that are 'sensors' located on the cell surface in order to capture external signals) as with objects $*$ that can stay only inside the *interstice* of a double membrane $[_i * [_i \quad]_i *]_i$.

By extending the boundary rules we have been able to model cellular dynamics with membranes, because we could modify the topology of a membrane structure. Namely, we model the moving of a membrane with its whole content into another membrane, as well as the generation of a 'tissue' by adhesion between membranes with the same label. Also, we modeled typical processes occurring in the cells, such as external signalling for receptor activation, adhesion between membranes by receptor affinity, production of internal signals, internal signalling for receptor activation, phagocytosis (with diapedesis and opsonization as particular cases), expulsion (as inverse of the phagocytosis), fusion, lysis, and apoptosis as a combination of affinity adhesion and lysis.

An important aspect in modelling biological reactions by rewriting rules is the rule application strategy. Nondeterministic maximally parallel way or other mathematically elegant strategies are not realistic. Percentages of reactivity assigned to rules would seem a better idea, but still several strategies can be performed. In fact, with regards to the BZ oscillator in [224], the same percentage of applications can give very different behaviours with rules applied in a different order. It seemed to us that both the order is essential and the reactivity parameters must depend on the concentration of certain objects in some membranes.

In this context, a novel perspective has been recently introduced by V. Manca's group [21, 149]. This is the idea of controlling the evolution of a (membrane) system by means of rules whose strength and application depend on the objects concentration. Rules are specified along with dynamical reactivities, called *reaction maps*. Every reactivity denotes the ability of the corresponding rule to compete against other rules in capturing part of the population on which the reaction is performed [23]. By going ahead in this perspective, every reactivity has been identified to a map that depends on the state of the system, and a strategy has been added for partitioning the objects in the system at every transition, by depending on the relative magnitude of every reactivity [22]. This new strategy of rule application was inspired by real 'metabolic reactions', and it seems to lead membrane computing towards interesting simulations of biological processes, such as representations of signal transduction networks and complex oscillations [24, 66, 25].

2.5.2 Applications

P systems as well as any biomolecular computing device, are at an initial development stage. On one hand, their simplicity does not limit their computational power and, in fact, numerous results of universality were proven. On the other hand, much research has still to be done to get them closer to the world of real (especially, but not only) biomolecular applications.

However, many recent, very promising works in this direction can be found in the book [43] and in the Proceedings of the first workshop on Theory and Applications of P Systems (TAPS), held in Timişoara in September 2005. Multiset processing in the compartmentalized framework provided by cell-like or tissue-like membrane structures turned out to be a widely applicable modelling technique in several domains, such as biology (for mechanosensitive channels, respiration in bacteria, photosynthesis), medicine (for immunological processes, signalling pathways), economics[10], linguistics, computer science (computer graphics, cryptography, approximate solutions to optimization problems). Computationally hard problems have been investigated through P systems (for example, in [93]), and problems coming from classical bioinformatics have provoked interest as well. For example, in [67] seven algorithms to find the maximum of a set of numbers were systematically investigated in terms of P systems, on the way to find better solving methods for the multiple sequence alignment problem.

In [21] we have focused our effort in addressing some theoretical and practical issues especially oriented to biomolecular computing—this applicative direction being followed also by other groups (see, for example, [7, 18]). There are some aspects, that are crucial in many studies of biomolecular processes, that the traditional formulations of P systems do not take into explicit account:

1. *dynamics* of biosystems. The dynamical behaviour of biomolecular processes has a major relevance in the description of the processes themselves. This means that two or more processes that terminate with identical configurations may move through completely different transitions. Think to the physiology of distinct human beings: we *know* that the biomolecular processes happening inside them inevitably move towards a well-determined terminal state. Rather, we would be interested in the *trajectory* followed by these processes, i.e., the set of intermediate states forming each trajectory.
2. process *evolution*. The halting of a P system tells that a computation has terminated successfully. For what we have written at the previous point, the terminal state is not a primary object of investigation in many biomolecular experiments. Rather, one should shift the focus on the computation during its "life", in the aim to investigate the strategies that a living organism puts into action to survive in certain environmental conditions.
3. *environmental energy* and *resources*. The resources available in the environment play a major role in the control of a biomolecular process. The existence in the environment of elements, which can act as catalysts or provide the energy needed for the biochemical elements to react, can radically change the

[10] Gh. Păun, R. Păun. *Membrane Computing as a Framework for Modeling Economic Processes*. Submitted.

nature of a process. In particular, an environment which periodically feeds the system with resources can transfer properties of periodicity to the system itself [20].

4. *locality* of the system control. Biomolecular processes are the result of many individual reactions, each one of those being formed by subprocesses whose extension is limited in space and in time. These subprocesses get the information on what is happening in the whole organism by their respective neighborhood. Though, they often exhibit a surprising overall coordination, and in this sense biomolecular processes are by all means *asynchronous*.

Clearly, these aspects are closely related one to the other: shifting the focus on the system dynamics means that less attention is payed to the final configuration of the system; meanwhile, the continuous control of the resources needed for the process to evolve is fundamental to drive the system dynamics along a specific trajectory. The environment itself has the role of a "supervisor" in the process control, since it becomes responsible of a sort of external input, whose effects in the system propagate via local reactions.

About the first point, usually P systems are intended to "consume" the available resources in a nondeterministic and maximally parallel way during the rewriting of symbols. In particular, the latter condition prevents the system to account for specific evolutions within a transition. In principle, one may try to increase the granularity of a P system in order to obtain fine-grained sequences of transitions, then consider the trajectories described by these sequences, and this description would be as accurate as fine the granularity of the P system is. In practice, it is likely that the desired granularity is obtained by adding auxiliary symbols or priority constraints in the system, to form (sometimes complex) priority relationships for the rewriting rules. As a matter of fact, P systems do not provide tools for controlling the resolution of the observation of intermediate states; they are better suited to model a process as a sequence of "snapshots", each one being taken when no more rewriting rules can be applied.

About the second point, there are cases in which a nonterminating process gives more insight than a terminating one. Nonterminating processes, in particular, are of key importance in the study of *periodicity* and *quasi-periodicity*, two aspects whose detection is important for understanding many biological processes. From this viewpoint, traditional types of P systems do not provide versatile tools to handle periodicity.

The question of resource availability is unavoidable in the study of biomolecular phenomena. P systems, in their native definition, did not take any energetic constraint into account. It was feasible for them to use indefinitely large amounts of resources to perform a computation, whereas this cannot happen in nature. Further types of P systems have been proposed in which energy is considered as a constraining factor (see, for example, [187]), although in those systems the environment is intended as a place to exchange resources and has not a central role.

Finally, P systems are traditionally organized in a way that their evolution is synchronous, i.e., a global clock triggers the production of new symbols inside all membranes. This aspect, that limits their versatility in modelling asynchronous phenomena, has been recently addressed in [36].

In [21] we considered a different perspective (for many aspects still in progress) according to which P systems were cast in a dynamical framework. In this perspective, we characterized transitions and space states of a discrete dynamical system by means of state transition dynamics [151]. Concepts of periodicity and quasi-periodicity of state transition dynamics were applied to P systems with boundary rules and dynamical environment (PBE systems); in particular, the *Brusselator* (a famous chemical reaction) was successfully simulated.

Overall, a new way to observe the evolution rules of a system reproducing a reaction was proposed. Indeed, since the application of every rule changes the relative amounts of reacting substances, it was imposed that such a relative amount influences the reactivity of the rules in a way that their application depends on the current substances concentration, as it normally happens in biochemical phenomena. A simulator applying evolution rules with this strategy (named *Psim*) was developed and used to simulate some processes [21]. In fact, an application of this strategy to problems so far described in terms of differential equations was proposed, beside a simulation with Psim of mechanisms typical of the *leukocyte selective recruitment process* activated against an inflammation.

Due to its complex nature and capability to rapidly adapt to the attack of infectious agents, the immune system may be considered a typical example of complex adaptive system [216]. A formal description has been developed in [75] about the phenomenon of leukocyte recruitment that plays a critical role in the immune response. In a next step, the leukocyte selective recruitment has been modelled (under some constraints that simplify the dynamics of the system) as a dynamical system of interactions between leukocytes and endothelial cells in [21], where an extended version of membrane systems turned out to be a very useful tool in the formal analysis of the recruitment process.

Leukocyte selective recruitment

The first response to an inflammatory process in a given organism activates a tissue-specific recruitment of leukocytes that relies on the complex functional interplay between the surface molecules that are designed for specialized functions. These molecules are differently expressed on leukocytes circulating in the blood and on endothelial cells covering the blood vessel (see Figure 2.7, which is taken from [75]).

Three major steps have been identified in the process of leukocyte recruitment: *i*) tethering-rolling of free-flowing white blood cells, *ii*) activation of them, and *iii*) arrest of their movement by means of their adherence to endothelial cells. After this arrest, *diapedesis* happens, that is, leukocytes from blood pass beyond endothelial cells into the tissue. If we call A the initial state with leukocytes quickly circulating into the blood, B the state of rolling, C the state of activation, and D the final state of adhesion, then the system has three big phases: $A \to B$ (by means of some receptor-receptor interactions), $B \to C$ (by means of some chemokine-receptor interactions), and $C \to D$ (by means of some receptor-receptor interactions).

Some molecules (called chemokines) are produced by the epithelium and by bacteria that have activated the inflammation process. Chemokines can bind with receptors expressed on a leukocyte, so producing signals inside it. These signals

generate on the leukocyte surface others and different receptors that, interacting with endothelial receptors, greatly slow down the speed of the cell, until it does not arrest (see Figure 2.7).

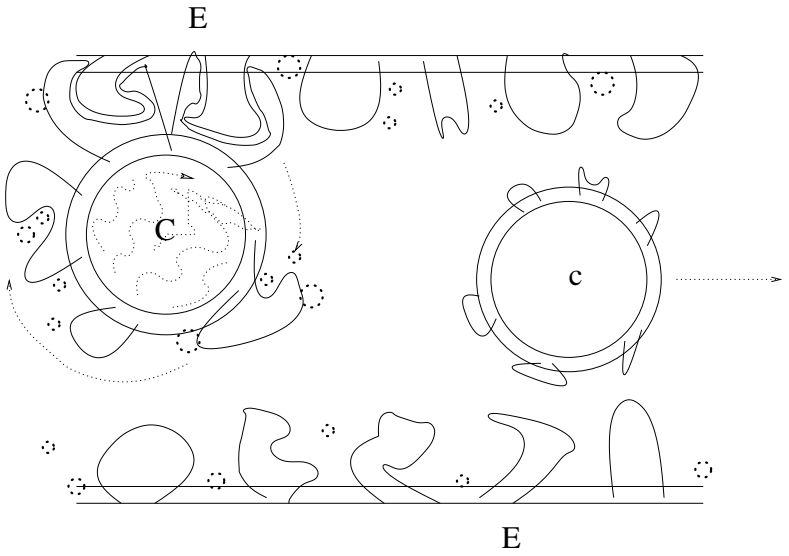


Fig. 2.7. Leukocyte cell C attacked by chemokines and endothelial receptors (the cells are leukocytes, and E is the epithelium).

For more details about such an immunological phenomenon and a membrane system describing it see [75], where, beside the traditional rules for communication and transformation of P systems [180], rules are allowed for the expression of receptors, for adhesion between membranes, and for the encapsulation of a membrane inside another membrane.

In this section we analyze with Psim a more simplified version of the system, where only the sound kind of specific leukocytes are present to attack the antigen, and only a production of symbols rules the dynamics (and in this first approximation receptor-receptor bonds are not explicitly represented) [21].

Our system is composed by two elementary membranes, with labels E and L. E-membrane represents the epithelium, which the bacterium enters and infects, and L-membrane represents the leukocyte which interplays with epithelium by means of production or transformation of symbols. Initially we have a symbol b inside the E-membrane.

The rules are displayed with respect to the following notation introduced in [21] for Psim. Each membrane is made up by three different regions: the inner part called *in*, the outer part called *out*, and the part across the physical membrane layer, called the *inter* region. This last part of the membrane usually contains receptors, we can imagine it as an intermediate zone between membrane inside and membrane outside. Let us name P the set $\{in, out, inter\}$, and T the set $O \times M$, where O is the alphabet (of objects) and M is the set of membrane labels of a membrane system. A rule has the form $h : v \to u$ where $h \in [0, 1]$, and v, u are

strings over $T \times P$. Namely, $k : (o, m, in) \to (o', m', out)(o'', m'', inter)$ means that the rule with reactivity k (that alters the uniform distribution of probability in the application of the rule), replaces the object o that is located inside the membrane m, with the object o' in the region external to the membrane m' and with o'' in the intermediate location of the membrane m''. In the following, for notational convenience, we write the string (c^n, m, t) instead of the string $(c, m, t)^n$, where $c \in O$, $m \in M$, $t \in P$, and $n \in \mathbb{N}$.

The simple process we model is the following. The antigen inside the epithelium produces externally substances called chemokines together with epithelial receptors, and internally copies of itself. Chemokines transform into internal signals inside the leukocyte (this simulates the activation of the leukocytes). These internal signals transform into leukocyte receptors in two steps. When a sufficient number of both epithelial and leukocyte receptors are present in the system, the elimination of bacterium is trigged by the production of a symbol v (this is an abstraction of the diapedesis phenomenon [75]).

We consider the alphabet $\{b, c, p, q_1, q_2, s, v\}$, and initially we have only the symbol b, representing the *replicating* bacterium, inside the E-membrane:

$$k_1 : (b, E, in) \to (bb, E, in).$$

During the process, b produces several symbols c (we fixed a rate of 3 copies for each copy of b), representing the chemokines, out of the membrane E, and more symbols p (2 copies for each b), representing the epithelial receptors, on the surface of E:

$$k_2 : (b, E, in) \to (c^3, E, out)(b, E, in)$$
$$k_3 : (b, E, in) \to (p^2, E, inter)(b, E, in).$$

The chemokines c out of the membranes transform into symbols s, representing internal signals, inside L-membrane, and s is transformed in q_2 (representing leukocyte receptors) 'slowly', that is by passing trough a transformation into 5 copies of q_1:

$$k_4 : (c, E, out) \to (s, L, in)$$
$$k_5 : (s, L, in) \to (q_1^5, L, inter)$$
$$k_6 : (q_1, L, inter) \to (q_2, L, inter)$$

Finally, the presence in the system of both p and q_2, representing epithelial and leukocyte receptors, actives the production of a symbol v inside the E-membrane, where each copy b is deleted by the presence of v:

$$k_7 : (p, E, inter)(q_2, L, inter) \to (v, E, in)$$
$$k_8 : (b, E, in)(v, E, in) \to (v, E, in).$$

Given this system, the point was to regulate the values of the reactivities $k_1, \ldots, k_8$ so that the application of the rules is modulated in the sound way which allows to neutralize the infection. As one can see in Figures 2.8 and 2.9, both taken from [21], once fixed $k_1 = 1$, $k_2 = 1$, $k_3 = 0.8$, $k_4 = 0.7$, $k_5 = 1$, $k_6 = 0.2$, $k_7 = 0.8$, $k_8 = 1$, the immune response works in a sound way in order to destroy all copies of a replicating b.

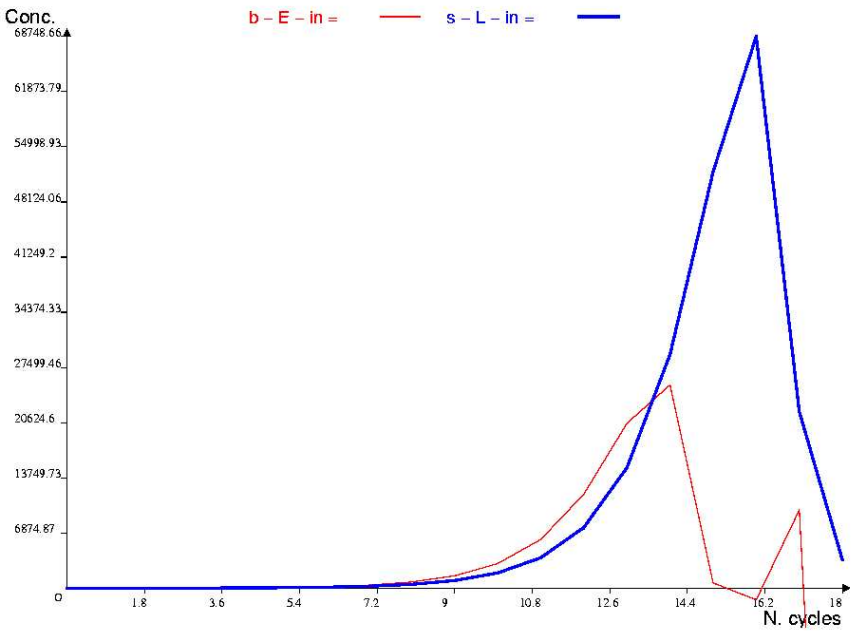


Fig. 2.8. Patterns of bacterial infection (red) and leukocyte internal signals (blue). Quantities of b inside E and s inside L after 18 applications of the rules and with a maximum capacity of membranes of 10^6.

In Figure 2.8 note how the decreasing of the infection (the quantity of b inside E) corresponds just to the increasing of the activation of leukocytes (quantity of s inside L), as it is well known from immunological studies [216]. Also, when the infection disappears then the production of s slows down quickly. Moreover, in Figure 2.9 one may see that the production of epithelial receptors is higher than the production of chemokines, while their behaviour following the pattern of b (infection) quantity, and this is confirmed from immunological studies as well [216].

For more details about the application strategy of the rules in each computational step of the simulations, see [21].

The content of this chapter may be respectively found along parts of the following papers, sometimes with more details and sometimes by a different approach:

- G. F., V. Manca, *An Algorithmic Analysis of DNA Structure (short version)*, Gh. Păun, C. Zandron, editors, Preliminary Proceedings of WMC 2002, Curtea de Argeş, Romania, August 19-23, pp 207–219, 2002.

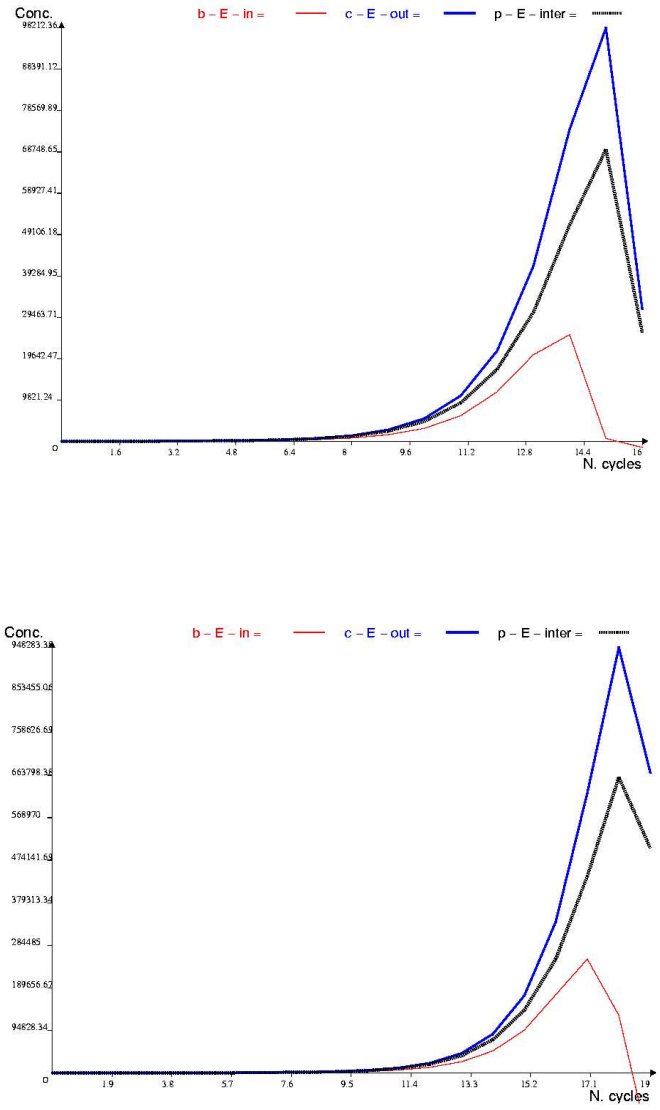


Fig. 2.9. Patterns of bacterial infection (red), chemokines (blue), and epithelial receptors (black). Respectively b, c, p quantities in: (top) 16 applications of the rules and 10^6 as maximum capacity of membranes, (bottom) 19 applications of the rules and 10^7 as maximum capacity of membranes.

- G. F., V. Manca, *An Algorithmic Analysis of DNA Structure*, Soft Computing, 9(10), pp 761–768, October 2005.

- G. F., M. Margenstern, *Computing by Floating Strings*, Proceedings of MeCBIC (Workshop on Membrane Computing and Biologically Inspired Process Calculi), Venice, Italy, July 9, 2006, ENTCS (Electronic Notes in Theoretical Computer Science), submitted.

- G. F., V. Manca, *A Membrane System for the Leukocyte Selective Recruitment*, C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, editors, Membrane Computing, Revised Papers of WMC 2003, LNCS 2933, Springer-Verlag, Heidelberg, Germany, pp 181-190, July 2004.

- L. Bianco, F. Fontana, G. F., V. Manca, *P Systems for Biological Dynamics*, Chapter 3 (invited) of G. Ciobanu, Gh. Păun, M. J. Pérez-Jiménez, editors, Applications of Membrane Computing, Natural Computing Series, Springer, pp 83-128, 2006.

3

Cross Pairing PCR

We can see only a short distance ahead,
but we can see plenty there that needs to be done.

Alan Mathison Turing[1]

The Polymerase Chain Reaction (PCR), which has allowed far more rapid amplification of DNA fragments than conventional molecular cloning, has been described as "being to genes what Gutenberg's printing press was to the written word". The idea underlying its methodology is simple, but its impact has been extraordinary. The first publication appeared in December 1985 [207], where the technique was used in an analysis of sickle cell anemia; a year later, the promise implicit in PCR amplification for large-scale sequencing efforts has been suggested in [171,211], and every year thereafter, the number of papers on PCR has increased tremendously [170]. In 1989, *Science* magazine selected PCR as the 'major scientific development' and Taq DNA Polymerase as the 'molecule of the year'. In 1993, Kary Mullis received the Nobel Prize for chemistry.

Because of PCR, 'insufficient DNA' has been no longer a limitation in molecular biology research or DNA-based diagnostic procedures. In fact, the full benefits of PCR in the emerging field of molecular diagnostic have yet to be realized [163]. Given the low frequency of sequence errors observed in such a technique, the speed and the sensitivity, several variants have been introduced, also for purpose of computation, such as whiplash PCR, whose idea was introduced in [210], and more recent quantitative (real time) PCR [121] and emulsion PCR [53].

In Section 3.3 we present a special type of PCR, called Cross Pairing PCR[2] (shortly XPCR), which was tested by laboratory experiments in several situations, all reported in Chapter 6. In fact, as the scientific methodology suggests, such a technique had to pass inevitably through an iterative process involving a close interaction between experimentation and computation.

[1] Computing machinery and intelligence. *Mind*, 59:433-460, 1950.

[2] Invention 'Cross Pairing Polymerase Chain Reaction' covered by a U.S. preliminary patent from Department of Commerce, Docket number: BUCH-001PRV.

As relevance of this method in the context of DNA computing, we show how it was basic in the biotechnological procedures which implemented an algorithm for generating a complete n-bit library (Section 4.1), an algorithm for selecting the strands containing a given substrand γ from an heterogeneous pool (Section 3.4), and an algorithm performing mutagenesis (Section 5.4). The experiments described in Chapter 6 confirmed the correctness and completeness of the biomolecular algorithms proposed, and their implementation resulted to be convenient with respect to the standard methods, for efficiency, speed and feasibility.

In general and abstract terms, XPCR is a method for performing a transformation on strings, that is essentially the splicing combinatorial mechanism in the original formulation introduced by Tom Head in [97]; more precisely, it implements a null context splicing rule (Definition 3.2). An analysis of PCR process beside some notions related to splicing rules are given respectively in the next sections as background for the description of XPCR method and XPCR based algorithms. An extraction procedure based on XPCR technique is proposed in Section 3.4, whereas in Section 3.5 one can find both basic concepts about involution codes, and some encoding problems related to the experiments.

All the bio operations we are going to consider are performed in a test tube containing many copies of each DNA sequence. In the following we say *pool* P the content of such a test tube; formally it is a multiset of strings, which is a set of strings having positive multiplicity. The *union* operation $\mathbf{P_1} \cup \mathbf{P_2}$ (also called *merge*) produces the pool resulting after a mix of the two pools P_1 and P_2.

There is a natural association between a DNA sequence and the number of its bases, that number is called the length of the sequence. Electrophoresis is a standard gel-based technique that selects DNA sequences with respect to their length. Its precision depends on the kind of gel, and also sequences that differs of one basis can be separated (technical details are reported in Subsection 6.2.2). We denote by $\mathbf{El_r}(\mathbf{P})$ the pool obtained from P after an electrophoresis where the sequences r long were selected, or, in other words, the pool containing all the sequences of P having length exactly r.

3.1 PCR

The Polymerase Chain Reaction is one of the most important and efficient tool in biotechnological manipulation and analysis of long DNA molecules. PCR exploits the remarkable natural function of the enzymes known as polymerases. These enzymes are present in all living things, and their job is to copy genetic material. Sometimes referred to as 'molecular photocopying', PCR can characterize, analyze, and synthesize any specific piece of DNA or RNA. It works even on extremely complicated mixtures, seeking out, identifying, and duplicating a particular bit of genetic material from blood, hair, or tissue specimens, from microbes, animals, or plants. The key to the process's automation has been the Taq polymerase. Taq is a nickname for *Thermus Aquaticus*, a bacterium that happily survives and reproduces in an environment that is lethal to other organisms: hot springs. That is why the organism's polymerase is perfectly at home in the rapidly fluctuating temperatures of automated PCR. Unlike other polymerases, the enzyme extracted from

Taq (and now made in commercial quantities by genetically engineered bacteria) is stable at high temperatures. The microbiologists who found these remarkable organisms decades ago, and then spent years studying their physiology and biochemistry, had no way of knowing how crucial their work would become to human health, to the forensic sciences, and to the economy (Section 5.5). Who would have thought a bacterium hanging out in a hot spring in Yellowstone National Park would spark a revolutionary new laboratory technique?

To copy DNA, polymerase requires two other components: a supply of the four nucleotide bases and an oligonucleotide called primer. DNA polymerases, whether from humans, bacteria, or viruses, cannot copy a chain of DNA without a short sequence of nucleotides to 'prime' the process, or get it started. In fact, the cell has another enzyme called primase that actually makes the first few nucleotides of the copy. Once the primer is made, the polymerase can take over making the rest of the new chain. In fact, starting from the primer, the polymerase can read a template strand and match it with complementary nucleotides very quickly. The result is two new helices in place of the first, each composed of one of the original strands plus its newly assembled complementary strand[3], as in Figure 3.1.

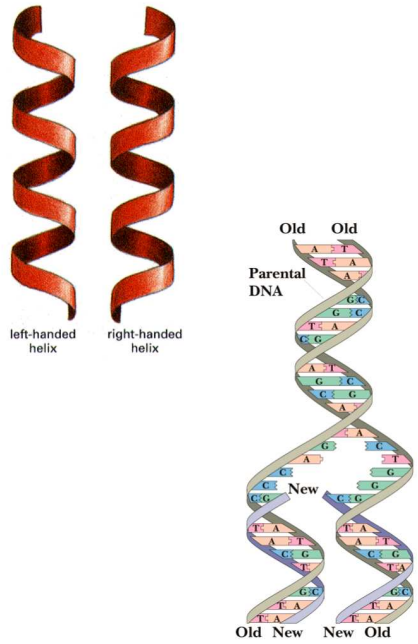


Fig. 3.1. Double helix 'semiconservative' autoduplication in the cell.

[3] In nature, most organisms copy their DNA in the same way. When any cell divides, enzymes called polymerases make a copy of all the DNA in each chromosome. The first step in this process is to "unzip" the two DNA chains of the double helix. As the two strands separate, DNA polymerase makes a copy using each strand as a template (Figure 3.1). The PCR mimics this process, only it does it in a test tube and by using only a $5'$-$3'$ writing polymerase.

For PCR, primers are the duplicates of nucleotide sequences occurring on each side of the piece of DNA of interest, which means that the exact order of the primers' nucleotides must already be known. These flanking sequences can be constructed in a biotechnologies laboratory, or purchased from commercial suppliers. From a computational point of view, the PCR procedure is thus based on: i) templates, ii) a copy rule applied to templates, iii) initial short strings (primers) that say where the copying process has to start. Polymerase enzyme 'writes' step by step, in the $5'$ - $3'$ direction, by coping (in complementary form) the bases of the template which drives the copy process, as in Figure 3.2, which is taken from [85].

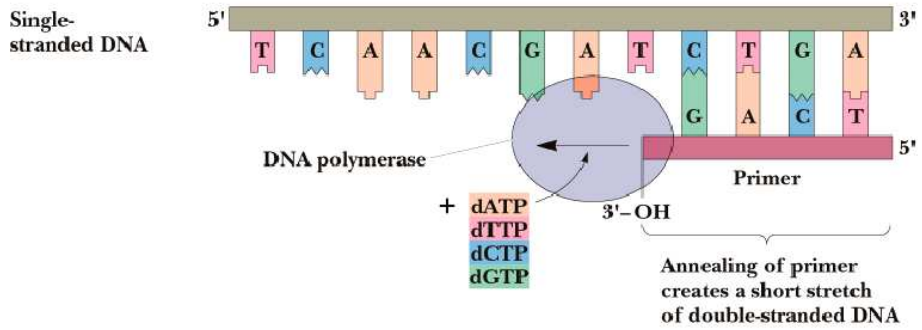


Fig. 3.2. Polymerase extension.

The three parts of the polymerase chain reaction are carried out in the same pool, but at different temperatures. The first part of the process separates the two DNA chains and this is done simply by heating the pool to 90-95 degrees centigrade; we indicate this operation on the pool P with $\mathbf{H}(\mathbf{P})$[4]. The primers cannot bind to the DNA strands at such a high temperature, so the pool is cooled to 55 degrees. At this temperature, the primers bind to their complementary portions along the DNA strands. We denote this step of (priming, or, more in general) cooling down, with $\mathbf{C}(\mathbf{P})$. The final step of the reaction makes a complete copy of the templates. The temperature of the pool is raised at around 75 degrees (to allow only primers to be annealed) in which the Taq polymerase works well. It begins adding nucleotides close to the primer and eventually makes a complementary copy of the template. If the template contains an A nucleotide, the enzyme adds a T nucleotide to the primer, if the template contains a G, it adds a C to the new chain, and so on to the end of the DNA strand. We denote this step by $\mathbf{Taq}(\mathbf{P})$, where a polymerase extension is performed in P by elongating all the annealed primers.

[4] This process is also called *denaturation*.

The three steps in the polymerase chain reaction (the separation of the strands, annealing the primer to the template, and the synthesis of new strands) take less than two minutes. They are described as instructions of an algorithm in the Table 3.1, where we call $\mathbf{PCR}_{(\alpha,\beta)}(\mathbf{P})$ a standard PCR with forward primer α and backward (reverse) primer $\overline{\beta}$ on the pool P, and we assume that all the ingredients necessary for PCR procedure (such as nucleotides, and polymerase; for technical details see Section 6.2.1) are present in the pool P. Such a procedure amplifies all the portions having as prefix the sequence α and as suffix the sequence β, since the sequences are considered in the $5'$ - $3'$ orientation.

$\mathbf{PCR}_{(\alpha,\beta)}(\mathbf{P})$
input P
begin
$P := H(P \cup \{\alpha, \overline{\beta}\})$;
$P := C(P)$;
$P := Taq(P)$
end
output P

Table 3.1. PCR Algorithm

The polymerase chain reaction performs many cycles where the basic step (Table 3.1) is repeated by the same primers and doubles each string beginning with the forward primer and ending with the backward one (see Figure 3.3, which is taken from [85]). An exponential number (with respect the number of cycles) of copies is obtained, and this amplification is generally exploited as extraction tool of strands having fixed prefix and suffix.

Given a double stranded molecule we describe the action of a PCR on it by means of a rewriting system. In particular, we analyze the different situations depending on the primers choice and we find a sort of general 'grammar' describing PCR phenomena. We indicate all the double structures involved in the process, sometimes we will say strings briefly, with symbols over the alphabet $\{a, b, c, d, f, S\}$. We set the starting double stranded molecule as $S = \frac{\alpha_1\alpha_2\alpha_3\alpha_4}{\overline{\alpha_1\alpha_2\alpha_3\alpha_4}}$ and consider the different products of a PCR process starting from (some copies of) S.

1. Traditional primers: $\alpha_1, \overline{\alpha_4}$. The rewriting rule is just

$$r : \ S \rightarrow S, S$$

where the comma between (floating) strings separates the different occurrences. If $S(0)$ is the initial number of copies of S in the pool, the number $S(p)$ of strings S after p applications of the rule r is $S(p) = 2\, S(p-1) = S(0)\, 2^p$, for $p \geq 1$.

2. Internal primers: $\alpha_2, \overline{\alpha_3}$.

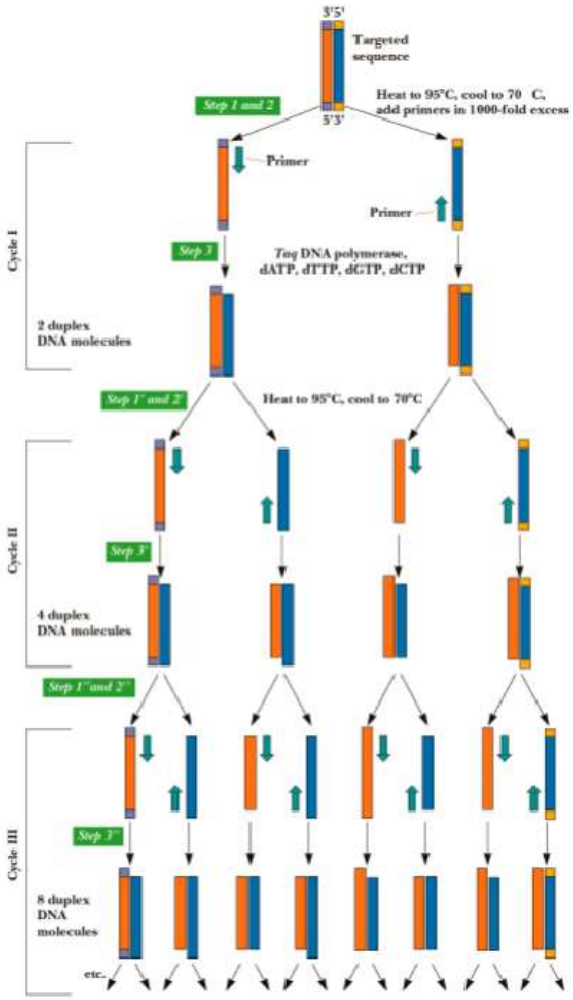


Fig. 3.3. PCR amplification.

If we set $a = \frac{\alpha_1\alpha_2\alpha_3\alpha_4}{\alpha_1\alpha_2\alpha_3}$, $b = \frac{\alpha_2\alpha_3}{\alpha_1\alpha_2\alpha_3}$, $c = \frac{\alpha_2\alpha_3}{\alpha_2\alpha_3}$, $d = \frac{\alpha_2\alpha_3\alpha_4}{\alpha_2\alpha_3}$, $f = \frac{\alpha_2\alpha_3\alpha_4}{\alpha_1\alpha_2\alpha_3\alpha_4}$, then the rewriting rules miming the PCR process with internal primers are

$$\begin{cases} S \to a,\ f \\ a \to a,\ b \\ b \to c,\ b \\ c \to c,\ c \\ d \to c,\ d \\ f \to f,\ d. \end{cases}$$

We assume that initially in the pool there are only copies of S, that is $a(0) = b(0) = c(0) = d(0) = e(0) = f(0) = 0$. Note that S disappears as soon as the process starts, the symbols a and f maintain a constant quantity equal to $S(0)$ after their generation in the first step: $a(p) = a(1) = S(0) = f(1) = f(p)$ with $p \geq 1$, the amount of symbols b and d increases linearly with the number of steps: $b(p) = (p-1)\, a(1) = (p-1)\, f(1) = d(p)$ for $p \geq 1$, and the double structure c is the amplified product. The increasing of its quantity can be written recursively as $c(p) = b(p-1)\, 2 + c(p-1)\, 2$, where the first term measures the production from the symbols b and d present in the previous step, and the second term quantifies the production from symbols c themselves.

3. Traditional and internal primers: $\alpha_1, \overline{\alpha_3}$ (the case $\alpha_2, \overline{\alpha_4}$ is analogous: in the system rules the symbol f would be replaced by a. Let $c = \frac{\alpha_2\alpha_3\alpha_4}{\overline{\alpha_2\alpha_3\alpha_4}}$).
In a sense, it is a simplified version of the previous system, because only from one (untraditional) primer the linear production is allowed.

$$\begin{cases} S \to S,\ a \\ a \to a,\ c \\ c \to c,\ c \end{cases}$$

where $a = \frac{\alpha_1\alpha_2\alpha_3\alpha_4}{\overline{\alpha_1\alpha_2\alpha_3}}$ and $c = \frac{\alpha_1\alpha_2\alpha_3}{\overline{\alpha_1\alpha_2\alpha_3}}$. As consequence of the traditional primer action $S(p) = S(0)$ for $p \in \mathbb{N}$, therefore $a(p) = S(0)\, p$. Whereas $c(p) = a(p-1) + 2c(p-1)$ with $p \geq 1$.

4. External primers: $\alpha_0\alpha_1, \overline{\alpha_4\alpha_5}$, leads a system analogous to the second case (with internal primers).

$$\begin{cases} S \to a,\ f \\ a \to a,\ b \\ b \to c,\ b \\ c \to c,\ c \\ d \to c,\ d \\ f \to f,\ d \end{cases}$$

where the symbols are analogously set as $a = \frac{\alpha_1\alpha_2\alpha_3\alpha_4}{\overline{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}}$, $b = \frac{\alpha_0\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}{\overline{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}}$, $c = \frac{\alpha_0\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}{\overline{\alpha_0\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}}$, $d = \frac{\alpha_0\alpha_1\alpha_2\alpha_3\alpha_4}{\overline{\alpha_0\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5}}$, $f = \frac{\alpha_0\alpha_1\alpha_2\alpha_3\alpha_4}{\overline{\alpha_1\alpha_2\alpha_3\alpha_4}}$.

5. Traditional and external primers: $\alpha_1, \overline{\alpha_4\alpha_5}$, leads to a system analogous to the third case.

In conclusion, a general PCR process amplifies portions of double stranded molecules, even elongating them by means of external primers. When un-traditional primers are used, then exponential amplification starts only after some steps and linear products are added in the pool. As a consequence, PCR process requires more material consumption and space occupation than in the standard case.

From a formal language theory viewpoint, the following system describes the whole process (see Figure 3.4):

$$\begin{cases} S \to a,\ f \\ a \to a,\ b \\ b \to c,\ b \\ c \to c,\ c \\ d \to c,\ d \\ f \to f,\ d \end{cases}$$

where every rule applies on an element, and replaces it with two elements, but there is no generation of languages. In fact, during the process the set of string-objects either remains $\{S\}$, by traditional primers, or remains $\{a, b, c, d, f\}$ after a fist step, where the symbol S has disappeared). Rather, the multiplicity associated to each object changes in each step, as often it happens in membrane computing, and the study of such dynamics is useful to analyze the amount of desired or redundant PCR products in the pool. This analysis would be useful in DNA computing, where the amount of DNA is a strict limit for the scale-up of many procedures. There exists a recent variant of PCR, called Real Time PCR [121], that measures the amount of amplified DNA. By such a method, the quantity of amplification is detected by a fluorescence added to the primers together with a quencher; it is released during the Taq polymerase extension so that finally the fluorescence results proportional to the amplified material.

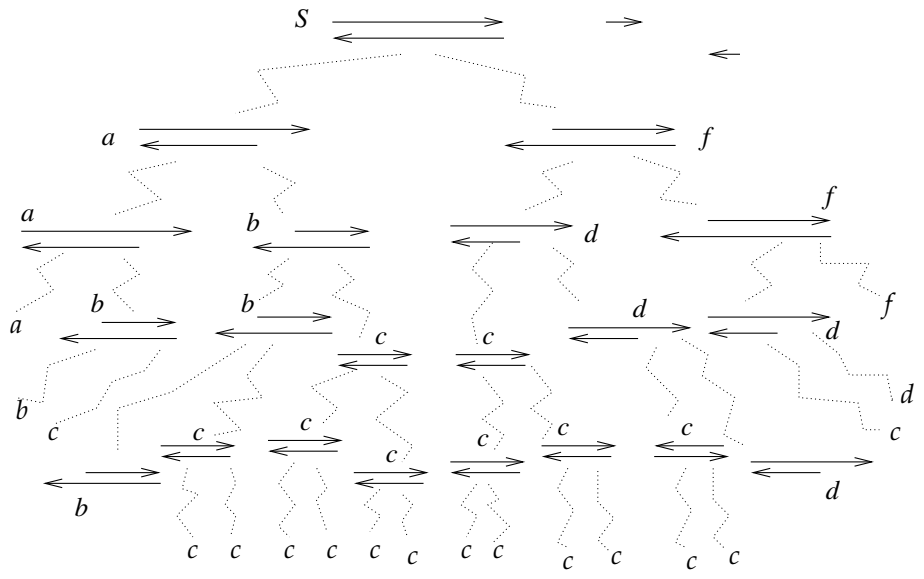


Fig. 3.4. PCR process.

As pointed out in [235], in the algorithms based on PCR one has to keep in mind that a restricted number of PCR operations are possible before to reach the limit of DNA quantity we can handle in a test tube. In our algorithms and experiments this information was considered in any step, and electrophoresis usually selected parts of previous amplifications.

3.1.1 PCR variants

In literature there exist many different variants of PCR. As examples we consider Whiplash PCR, that has been used as computational tool in [94,210,236,134,197], and Emulsion PCR [91, 53], that has been exploited for very recent sequencing technologies in [219].

Whiplash PCR originally used a single molecule and its hairpin formation to implement a finite state machine by polymerase extension [94], and then it was combined with combinatorial assembly PCR to generate structured libraries and to solve several NP-complete problems in a constant number of biosteps [236]. WPCR utilizes the facts that a single stranded DNA can anneal to a portion of itself (by forming a hairpin structure) and that polymerase extension can perform a task of reading out data written on the DNA sequence itself. The single stranded molecule is composed by an input segment on the $5'$-end, followed by a μ-formula (a boolean formula with each variable occurring once) segment (program), followed by a spacer, and finally by a 'head' on the $3'$-end that moves and performs the computation. The head anneals to complementary subsequences in the input and formula alternatively, exactly in the same way the transitions were implemented in [210], where the change from a $state_i$ to the next state $state'_i$ of an automaton was described as in Figure 3.5.

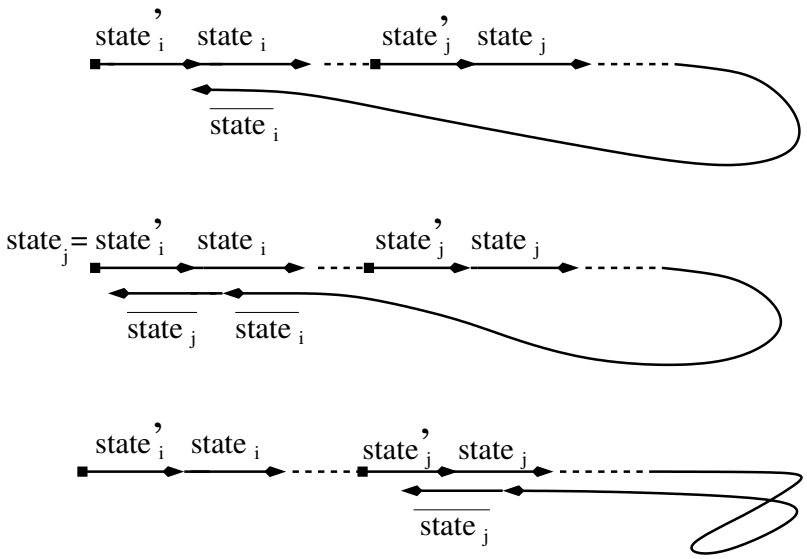


Fig. 3.5. Current state_i is attached to form an hairpin structure and its extension is performed by polymerase action. After denaturation the computation advances with the new current state_j at the $3'$-end.

On the experimental front, in [94] was demonstrated how to perform the polymerization stop reaction, and two successive head transitions of a very simple version of (very short) self-acting molecule were observed, but little is known about the capability of whiplash to perform computations in practice. In fact, from the

point of view of DNA Computing, Whiplash PCR is a nice model of computation with the advantage of performing single-tube computation of many programs in parallel, but it faces serious problems and limitations that need to be addressed experimentally.

An obvious question is how large a single-stranded molecule can be, and how longer it can grow during the computation without breaking. The answer could pose limits on the size of problems that the method can handle. A more sever problem would be generated by formations of loops in different parts of the sequence. Another critical problem is the possibility of different molecules interacting. In particular, it is necessary to determine at what concentration molecules start to interact with each other at a significant frequency (forming aggregates observable in the gel) and how these interactions scale with the size and number of molecules used. Allowable levels of concentration in order to minimize interactions could very likely be dependent on the size of molecules. They should be investigated so that the crucial volume requirements of the method could be determined.

Recent progress in WPCR techniques can be found in [134] where implementation of input/output interface and the multiple transition steps on solid phase are reported. In fact, in the implementation of WPCR with I/O interface, it is of great importance to keep the independence of each molecular machine, but at the same time it is also necessary to prevent an intermolecular reaction, and this was done by adopting a solid phase method [243].

In PNA-mediated Whiplash PCR (PWPCR) [196], an autonomous molecular computation is implemented by the recursive polymerase extension of a mixture of DNA hairpins. Like other methods based on exhaustive search, however, application to problem instances of realistic size is prevented by the exponential scaling of the solution space. The tendency of evolving populations to minimize the sampling of large, low fitness basins suggests that a DNA-based evolutionary approach might be an effective alternative to exhaustive search, and in [197] PWPCR has been modified to support the evolution of a population of finite state machines. Moreover, an in vitro algorithm for applying this architecture to evolve approximate solutions to instances of NP-complete problems, such as Hamiltonian Paths, is described in detail.

Together with in situ rolling circle amplification (RCA), bridge polymerase chain reaction [145] and picotiter PCR, Emulsion PCR is a technology performing multiplex amplification while maintaining spatial clustering of identical molecules to amplify. An optimized version of it has used to robustly and reproducibly amplify complex libraries for a new low-cost and rapid DNA sequencing technology [219]. Such a technique is the actual implementation of a membrane system with many elementary membranes (inside the skin membrane representing the test tube) where different PCRs are carried on.

The idea underlying the Emulsion PCR was introduced in [226], and the technique was then improved in [91], where a compartmentalized selfreplication reaction (CSR) was suggested as a strategy to isolate individual polymerase variants in separate compartments, in order to select polymerases with novel and useful properties. Provided that in each compartment one has genes and corresponding primers, each modified polymerase replicates only its own encoding gene to the exclusion of those in other compartments. Consequently, only genes encoding ac-

tive polymerases are replicated, whereas inactive variants fail to amplify their own genes. Among differentially active variants, the more active can be expected to produce proportionally more 'offspring'. Such reactions took place in individual aqueous compartments of a water-in-oil emulsion that are stable for prolonged periods at temperatures exceeding 90°C. A water-in-oil emulsion permits millions of noninteracting amplifications within a milliliter-scale volume.

Compartments have average diameters of 15 μm and they are proved to be heat-stable. Once formed, they do not permit exchange of macromolecules like DNA or protein. Presumably, their large molecular weight and electric charge prevent their diffusion across the droplets surface, even at high temperatures, and the droplets work as microreactors [91]. Small molecules instead, including the negatively charged dNTP (free nucleotides necessary to perform PCRs), can cross the surfaces and stay between the compartments during thermocycling. By increasing the oil-in-water ratio of the emulsion though, dNTP diffusion could be controlled [91], and in general, controlled communication between compartments offers prospects for the extension of CSR technology, while the model seeming to fit exactly a membrane system.

Emulsion PCR was used in [53] as extraction method, to detect a certain string in an initial population of DNA molecules, and to quantify the fraction of some kind of DNA molecules in that population. Such a technique resulted successful to analyze a large number of molecules independently while retaining a high signal-to-noise ratio. Amplification products of individual compartments in [53, 219] are captured via inclusion of 1-μm paramagnetic beads bearing one of the PCR primers (beads having been previously coated with streptavidin and bound to biotinylated oligonucleotides). Any single bead bears thousands of single-stranded copies of the same PCR product, whereas different beads bear the products of different compartmentalized PCR reactions (see Figure 3.6, which is taken from [219]).

Increasing template concentration in an emulsion PCR reaction boosts the fraction of amplified beads at the cost of greater non-clonality. The beads generated by emulsion PCR have highly desirable characteristics: high signal density, geometric uniformity, strong feature separation, and a size that is small but still resolvable by inexpensive optics.

Fig. 3.6. An aqueous mix containing all the necessary components for PCR plus primer-bound beads and template DNA are stirred together with an oil/detergent mix to create microemulsions. The microemulsions are temperature-cycled as in a conventional PCR. If a DNA template and a bead are present together in a single aqueous compartment, the bead-bound oligos act as primers for amplification.

This is a quite sophisticated and still experimental technique, even because the use of beads requires very specialized instruments and laboratories. In Section 3.3 we propose another variant of the PCR technique, but before to get under way let us review the theoretical context of the splicing rules, since it implements a particular combinatorial rule called null context splicing.

3.2 Splicing rules

The splicing systems proposed by Head in [97] represent a significant body of literature; we recall here some definitions as introduced originally.

Over a finite alphabet A, a *rule set* R is a set of strings $p\#q\$u\#v$ where p, q, u, v are strings of $A^\star$ and $\#$, $\$$ special symbols. The combination of an *initial language* I contained in $A^\star$ with a rule set R yields a *splicing system* $G = (A, I, R)$. This triple is regarded as a formal device, inspired by cut and paste phenomena on DNA molecules under the action of restriction and ligase enzymes, that generates from I a language $L(G)$ that is defined recursively. $L(G)$ is the smallest language in $A^\star$ that contains I and has the following closure property with respect to R: for each pair of strings $wpqx$ and $yuvz$ in $L(G)$, where w, x, y, z are in $A^\star$ and $p\#q\$u\#v$ in R, $wpvz$ is also in $L(G)$, and we say that it is obtained by splicing from the two strings.

Definition 3.1. *A language L is said to be a splicing language if there exists a splicing system $G = (A, I, R)$ for which $L = L(G)$.*

An element $p\#q\$u\#v$ of the rule set is called *splicing rule* and it recombines two strings along the two different *sites* $p\#q$ and $u\#v$ as it follows:

$$w\,p\,q\,x,\;\; y\,u\,v\,z \;\longrightarrow\; w\,p\,v\,z$$

A "symmetric" version of such a rule produces both the strings $wpvz$ and $yuqx$ when applied on $wpqx$ and $yuvz$. The symmetric splicing rules model the behaviour of enzymes that cut and glue DNA molecules. In fact, the restriction enzymes will bind to DNA at a specific recognition site and then cleave DNA mostly within, but sometimes outside, of this recognition site, while the ligase enzymes glue together two molecules wherever it finds nicks between symbol-nucleotides, as in Figure 3.7 which is taken from [69]. This type of cut and paste activity provides the basis for the technology of gene splicing [99].

Finite splicing systems are those having only finitely many rules and finitely many initial strings. Languages generated by such systems are necessarily regular, but not all regular languages can be so generated [146,147]. These systems generate just a restricted subclass of the class of regular languages [103], for example, they are not able to generate the regular languages $a^*ba^*ba^*$ and $(aa)^*$.

If one considers particular splicing rules $p\#q\$u\#v$ where (namely) $p = u$ and $qv = \lambda$, then one obtains the following kind of rule, also called *one sided context* because tests context on only one side [99].

Definition 3.2. *The null context splicing rules are of type*

$$\phi\,\gamma\,\psi,\ \ \delta\,\gamma\,\eta \quad \xrightarrow{r_\gamma} \quad \phi\,\gamma\,\eta,\ \ \delta\,\gamma\,\psi$$

where ϕ, γ, ψ, δ, η *are strings on the considered alphabet.*

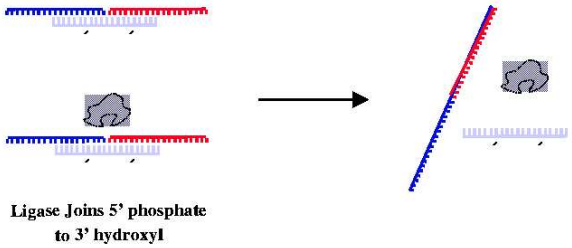


Fig. 3.7. DNA ligase action.

A *null context splicing system* $G = (A, I, R)$, called NCH system in [99], over an alphabet A, is a finite splicing system where R contains only null context splicing rules. The very elementary structure of NCH splicing languages is due to the fact that sites which have been used for recombination are still there. This need not be the case with sets of two or more enzymes that have non-null context.

The language $L(G) = L(A, I, R)$ generated by G is the smallest language L in $A^\star$ that contains I and has the property that whenever strings upx and ypz are in L, with r_p in R, the strings upz and yrx are also in L. It is clear indeed that the definition of constant, as introduced by M. P. Schutzenberger in [213] and recalled in [198], is a valuable conceptual tool for splicing theory:

Definition 3.3. *A string* w *is a constant for a language* L *if, whenever strings* xwy *and* uwv *are in* L*, the strings* xwv *and* uwy *are also in* L*.*

A language L is called a *null context splicing language* if there exists a null context splicing system $G = (A, I, R)$ for which $L = L(G)$. The null context splicing languages were introduced in [97], where the author has observed that, from a known theorem proposed in [51], it follows they are the strictly locally testable (SLT) languages as defined originally [167]:

Definition 3.4. *A language* L *over an alphabet* A *is called strictly locally testable if there exist finite sets* $P, S, K \subset A^\star$ *such that* $L = (PA^\star \cap \overline{A^\star S})\backslash A^\star K A^\star$*.*

In other words, a language generated by null context splicing rules starting from a finite set of sequences is characterized by having words with a finite number of prefixes and of suffixes, and that may not contain a finite number of subwords. More characterizations about null context splicing languages and a subclass called *simple H systems* can be found in [100] and in [164], respectively.

The recombination of strings having a specific common substring γ is performed in nature by some restriction enzymes [100]. They cleave the double helix at the

end of γ wherever they find the sequence, in such a way that (after a cut) a common overhang γ allows the recombinations of different DNA molecule along it.

This operation is very useful in biological situations (as mutagenesis) and in DNA Computing, but the limited number of enzymes stops the possible applications and the scale up to computations solving problems with high dimension, because by using specific enzymes we are able to perform only few different cuts. On the contrary, the procedure $XPCR_\gamma$ implementing the rule r_γ has no limit regarding the choice of (the primer) γ, and it was tested working as expected [72] with different string values for γ.

3.3 XPCR procedure

Repertoire selection methods have proven to be an effective means to obtain biopolymers with desired properties [91]. In this section we present the XPCR technique, introduced as tool of extraction in [72] (from which Figures 3.8, 3.9, 3.10, and 3.11 are taken). It is a variant of the standard PCR, where two dsDNA molecules and two primers are used in such a way that one primer hybridizes with one single strand of a DNA molecule, and the other primer with one single strand of the other DNA molecule. A similar idea, but in a very different context, was considered in [145].

Starting from a heterogeneous pool of DNA double strands sharing a common prefix α and a common suffix β, and given a specified string γ, by means of $XPCR_\gamma$, we can recombine all the strands of the pool that contain γ as substring.

The $XPCR_\gamma$ procedure is described by the following four steps.

input a pool P of strings having α as prefix and β as suffix

1. **split** P into P_1 and P_2 (with the same approximate size);
2. **perform** $P_1 := PCR_{(\alpha,\gamma)}(P_1)$ **and** $P_2 := PCR_{(\gamma,\beta)}(P_2)$ (*cutting step*[5], Figure 3.8);

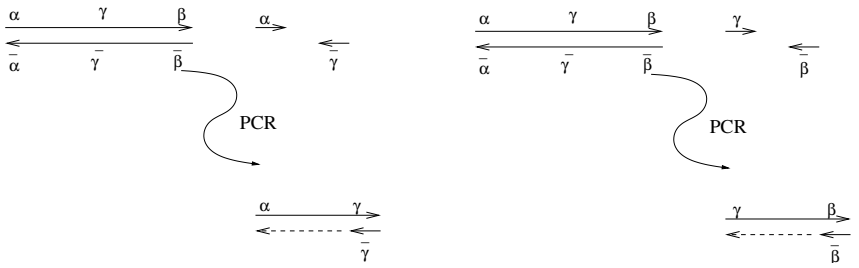


Fig. 3.8. Cutting step of XPCR$_\gamma$.

[5] In order to have the parallelism of these PCRs the encodings of primers must be such that their melting temperatures (see Section 6.1) are approximately the same.

3. **mix** the two pools resulting from the previous step in a new pool $P := P_1 \cup P_2$;
4. **perform** $P := PCR_{(\alpha,\beta)}(P)$ (*recombination step*, see Figure 3.9);

output the pool P resulting from the previous step.

After the cutting step we find in the test tubes an exponential amplification of the dsDNA $\alpha \dots \gamma$ and $\gamma \dots \beta$, respectively (see Figure 3.8), that are shorter than the initial molecules (linearly amplified products keep the initial length). In the recombination step, left parts $\alpha \cdots \gamma$ and right parts $\gamma \cdots \beta$ of the sequences of the pool having γ as subsequence are recombined in all possible ways, regardless to the specificity of the sequences between α and γ, or γ and β. Therefore, not only the whole sequences containing γ are restored but also new sequences are generated by recombination (see Figure 3.9).

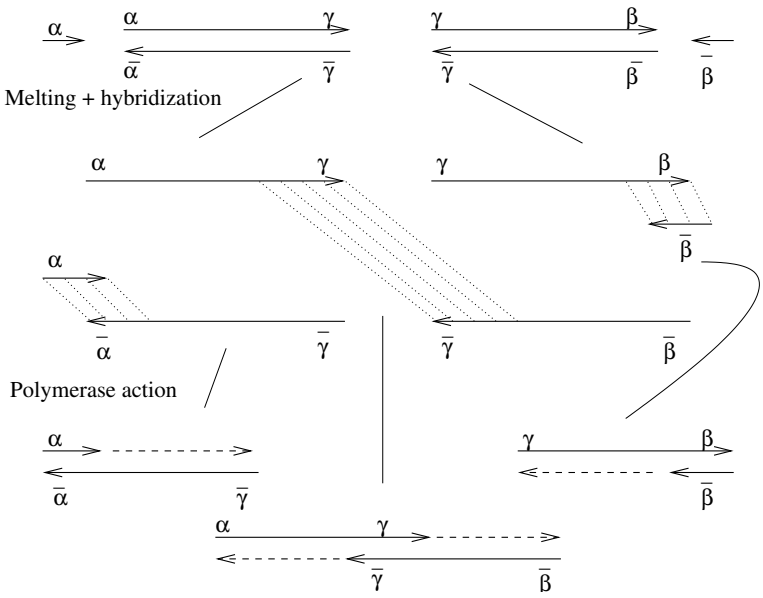


Fig. 3.9. Recombination basic step of XPCR$_\gamma$.

If one puts in a test tube many copies of $\alpha \dots \gamma$ strands and $\gamma \dots \beta$ strands (that finish and start respectively with a same substring γ), and two primers α and $\overline{\beta}$, then PCR performs the process as one can see in Figure 3.9, where the sequences $\alpha \dots \gamma \dots \beta$ are firstly generated, by a sort of overlapping juxtaposition of the two initial strands, and then amplified. This step has been verified in laboratory with three sizes of γ (229, 95, 15 bp, see Section 6.3).

As one can see in Figure 3.9, differently from standard PCR the primers hybridize with single strands of two different dsDNA molecules, so liberating the respective partners in each molecule. At this point, these single strands can hybridize each other by means of their (reversed) complementary parts γ and $\overline{\gamma}$, and the polymerase uses the single strand components of this structure as templates in order to complete the double string.

In the pictures related to each step of XPCR we do not mention the products given by secondary linear amplification (in the cutting step), because ignoring them will not affect the validity of the procedure. In the recombination step there is no production of redundant material, because only the recombined (long) strings are amplified (see Figure 3.10) and the quantity of short pieces remain constant. In fact, one of the strands of the initial short pieces in every step is the template for the generation of another short piece, and the two generated short pieces will join to form a long string including γ.

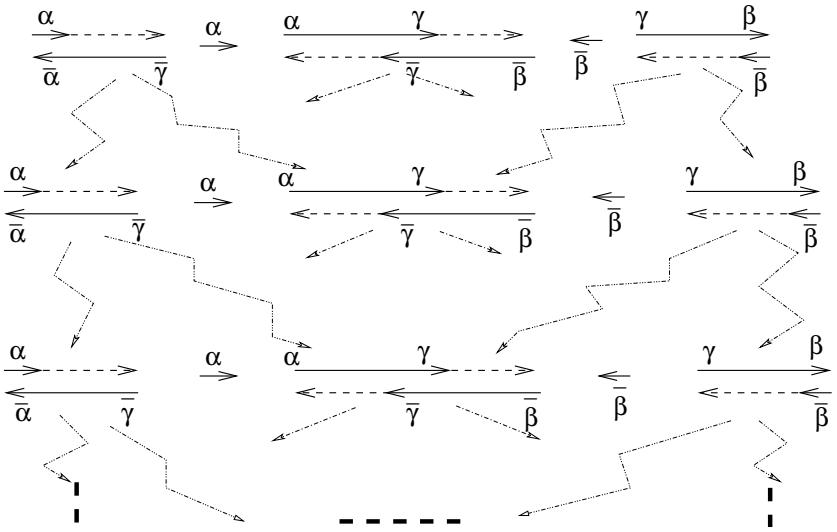


Fig. 3.10. Amplification of XPCR. Long strands in the middle are seeds of exponential amplifications.

In this form of the algorithm (that we have chosen for simplicity of explanation), during the recombination step also all the sequences of the pool that do not contain γ are amplified. In order to avoid this production, it is enough performing an electrophoresis after the cutting step to eliminate all the sequences longer than the γ-prefixed or γ-suffixed strings, as it was done by implementing the extraction algorithm presented in Section 3.4.

Finally we note that procedure $XPCR_\gamma$ implements a particular version of null context splicing rule with common substring γ, where strings are assumed to share a common prefix and a common suffix (both ϕ and δ start with a certain prefix, and both ψ and η end with a certain suffix) and strings obtained after their recombination are added to the strings available before it

$$\phi\,\gamma\,\psi,\ \ \delta\,\gamma\,\eta \quad \xrightarrow{r_\gamma} \quad \phi\,\gamma\,\eta,\ \ \delta\,\gamma\,\psi,\ \ \phi\,\gamma\,\psi,\ \ \delta\,\gamma\,\eta.$$

A version of XPCR that purely implements a null context splicing rule (without maintaining copies of the premise strings) is proposed in [150]. Whereas a generalized version of XPCR using external primers can be used for concatenating strings

in the following way. Suppose to have a pool P_1 of strings having prefix α and suffix ξ, and a pool P_2 of strings having prefix ζ and suffix β. Given a (primer) string γ, we can generate a kind of language concatenation $P_1\gamma P_2 = \{\delta\gamma\eta \mid \delta \in P_1,\ \eta \in P_2\}$ with the algorithm in Table 3.2. If $\xi = \zeta$, then $P_1\,P_2$ is obtained simply performing $PCR_{(\alpha,\beta)}(P_1 \cup P_2)$, that is the recombination step of $XPCR_\xi$.

$P_1 := PCR_{(\alpha,\xi\gamma)}(P_1)$ **and** $P_2 := PCR_{(\gamma\zeta,\beta)}(P_2)$;
$P_1\gamma P_2 := PCR_{(\alpha,\beta)}(P_1 \cup P_2)$.

Table 3.2. Concatenating by PCR

A differently arranged concatenation of sequences by XPCR was introduced and tested experimentally in [150].

3.4 An extraction algorithm

The extraction of DNA strands including a given sequence of bases is a crucial step in both DNA computing and biotechnology. The extraction step of the Adleman-Lipton paradigm remains its critical point. The initial methods used membrane filters for separating by Watson-Crick complementarity the strands containing given substrands from the other ones. With a similar strategy the biotin-avidin affinity allowed to select some strands by means of the complementary substrands bond to beads, and this method had efficiency $88 \pm 3\%$ [123]. In particular, as it is reported in [141], if we call p the percentage of sound extractions (extracting each of the required DNA strands is equally likely) and we assume that for each distinct string s in a test tube there are 10^l ($l = 13$ proposed by Adleman in [1]) copies of s, then no matter how large l is and no matter how close to 1 p is, there always exists a class of 3-SAT problems such that DNA computing error must occur.

An alternative extracting method used hairpin formations of single strands as test of inconsistent assignments of variables [209], and this kind of computation was combined with Whiplash PCR for simulating finite state machine [94]. More recently, microfluid devices for separating strands were introduced in [228], and an automated mix of thermocycler and gel-electrophoresis was developed in [28] in order to extract the strands encoding the satisfying assignments of a formula from a large pool of molecules. Finally, the idea of using PCR as a "very elegant and effective detection method" to check the existence of a solution was considered in [203], where a theoretic method 'blocking' the wrong sequences with PNA strands is proposed.

We address the following problem: given a specified sequence γ of bases, and an input pool P of different dsDNA molecules with a same length and sharing a common prefix and suffix, we want to produce an output pool P' where only the strands which include the given sequence γ are represented. Here we present an

algorithm which performs a specified extraction from a given pool of DNA double stranded molecules by using XPCR combined with gel-electrophoresis.

Let us start with a pool P which is constituted by dsDNA molecules having length n, α as prefix β as suffix. Given a string γ, let us assume that P is γ-*invariant*, that is, either γ does not occur at the same position in different strands of P, or if it is not the case, then $\alpha\tau_1\gamma\tau_2\beta, \alpha\tau_3\gamma\tau_4\beta \in P$ implies that $\alpha\tau_1\gamma\tau_4\beta, \alpha\tau_3\gamma\tau_2\beta \in P$. The hypothesis of a common prefix and suffix and the γ-*invariance* of the pool are not restrictive assumptions in the context of DNA computing. As usual, we call γ-*superstrand* any strand which includes a γ as substrand. The following procedure gives as output a pool P' containing all the γ-superstrands of P.

Algorithm

input a pool P as in the hypothesis

1. **perform** $P := PCR_{(\alpha,\gamma)}(P)$;
2. **perform** $P := PCR_{(\gamma,\beta)}(P)$;
3. **select** by gel-electrophoresis the short strands of lengths l_1, l_2 such that $l_1 + l_2 - l = n$, where l is the length of γ, $P' = El_{l_1}(P) \cup El_{l_2}(P)$;
4. **perform** $P' := PCR_{(\alpha,\beta)}(P')$;
5. **select** by gel-electrophoresis the n-long strands $P' := El_n(P')$;

output the pool P' obtained by the previous step.

Note that after the first gel-electrophoresis only the strings $\alpha \dots \gamma$ and $\gamma \dots \beta$ are selected, while the last one select all the longest strands in the current pool, that are just all the γ-superstrands of P exponentially amplified in the previous step (see Figure 3.11). The successful experiment implementing this algorithm in different cases is reported in Section 6.3.

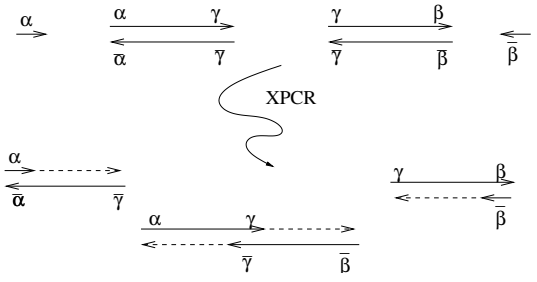


Fig. 3.11. XPCR-based step.

The advantage of XPCR based extraction method with respect to methods from literature is that it is rapid, cheap, and easy to implement. In fact, it requires only PCR and electrophoresis techniques, both standard and very efficient methodologies. On the other hand, as a disadvantage we point out its sensitivity, in the sense that any occasional error would be exponentially amplified. That is, with this procedure it is easier to find false positive than false negative.

In principle, operations as consecutive DNA extraction from a given pool by means of XPCR should work correctly, but problems could arise if the encoding is not robust enough for avoiding unexpected annealing. In order to avoid undesirable hybridizations between words the language involved by the DNA operations has to satisfy certain coding properties, and this is a requirement to investigate before any experiment. In fact, DNA strand design is one of the hardest subdisciplines of DNA computing.

3.5 DNA involution codes

The computation language of a DNA-based system consists of all the words (DNA strands) that can appear in any computation step of the system. Common approach to avoid undesired intermolecular and intra-molecular hybridizations has been to use the Hamming distance as a measure for uniqueness of strands [214]. Also, theoretical methods for constructing coded DNA languages were investigated [114] and decidability of desired properties were addressed as well [106].

We give an overview of the main characterizations of *involution codes*, as properties which ensure that the words in these languages will not form undesirable bonds when used in DNA computations. Given an involution $h : \Sigma^\star \to \Sigma^\star$ over a finite alphabet Σ, the involution codes *h-infix*, *h-comma-free* and *h-k-codes* intended for avoiding intermolecular hybridizations, and *h-subword-k-codes* intended for avoiding intra-molecular hybridizations, are presented in the following.

An *involution* h is a mapping such that h^2 equals the identity mapping, thus it is bijective and $h = h^{-1}$. It is called *morphic* if it is a morphism, and *antimorphic* when $h(uv) = h(v)h(u)$ for any strings u, v on the given alphabet. A *code* X is a subset of Σ^+ satisfying the property that, for every word $w \in X^+$, there is a unique sequence $v_1, v_2, \ldots, v_n$ of words in X such that $w = v_1 v_2 \cdots v_n$. The following definitions are basic in the study of DNA coded languages, where X is a finite code and $h : X^\star \to X^\star$ is a morphic or antimorphic involution.

Definition 3.5. *X is h-infix if $A^\star h(X) A^+ \cap X = \emptyset$ and $A^\star h(X) A^+ \cap X = \emptyset$.*

In other words, if X is a h-infix involution code, an element of $h(X)$ may not be a proper substring of an other element of X (see Figure 3.12). A stronger condition for X is the constraint to be an h-comma-free code.

Definition 3.6. *X is h-comma-free if $X^2 \cap A^+ h(X) A^+ = \emptyset$.*

In fact, by contradiction, if an element of $h(X)$ is a proper substring of another one in X, then it is going to be an element of $X^2 \cap A^+ h(X) A^+$. Conversely, over the alphabet $\{a, b\}$, the h-infix involution code $X = \{ab\}$ with $h(a) = b$ is not

h-comma-free. In fact, $X^2 = \{abab\}$ and $h(X) = \{ba\}$, therefore the h-comma-freedom is a strictly stronger condition. On the other hand, h-infix codes are closed under arbitrary concatenation while h-comma-free codes do not.

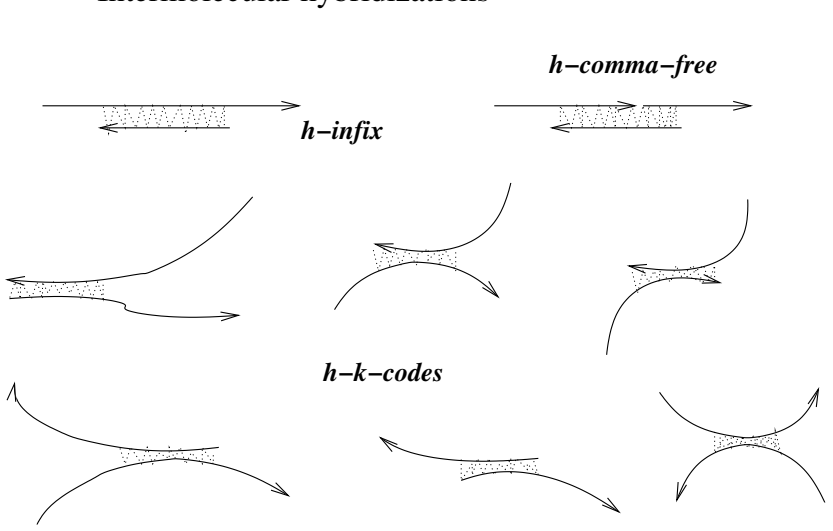


Fig. 3.12. Forbidden cases for h-infix, h-comma free, and h-k-codes.

In this chapter we have used languages X with words of a fixed equal length. They are 'almost' h-infix, in the sense that it holds $X^2 \cap A^+h(X^2)A^+ = \emptyset$ (see Lemma 1 in [113]).

To present a last property avoiding the intermolecular hybridization we recall that $Sub_k(X)$ is the set of all the k long subword of words in X.

Definition 3.7. *X is h-k-code if for some $k > 0$ $Sub_k(X) \cap Sub_k(h(X)) = \emptyset$.*

All the forbidden cases of the above involution codes inspired by the problems arising from the intermolecular hybridizations are illustrated in Figure 3.12.

The intra-molecular hybridizations generating (undesired) hairpin formations are not a minor problem for DNA computations, and different versions of involution codes were defined to avoid them. The point is bounding in a suitable way the length of the portions complementary to each other lying on a sequence and their distance along the sequence, see Figure 3.13.

Definition 3.8. *For all sequences $u \in A^\star$, X is a*

- *h-subword-k-code if, for $|u| = k$ and $i \geq 1$, $A^\star uA^i h(u)A^\star \cap X = \emptyset$*
- *h-subword-$k-m$-code if, for $|u| = k$ and $1 \leq i \leq m$, $A^\star uA^i h(u)A^\star \cap X = \emptyset$*
- *$h(k, m_1, m_2)$-subword-complaint code if, for $|u| > k$ and $m_1 \leq m \leq m_2$, $A^\star uA^m h(u)A^\star \cap X = \emptyset$.*

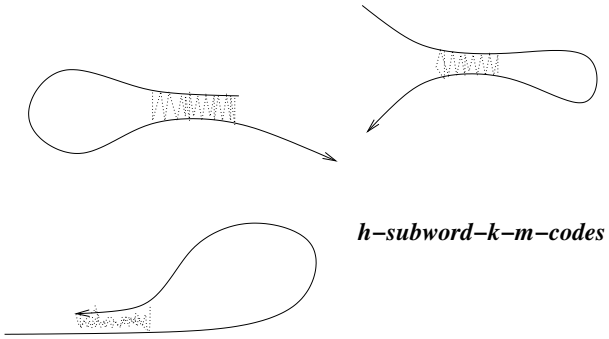


Fig. 3.13. Hairpin formations avoided by h-subword-k-m-codes.

The conditions expressed in Definition 3.8 are necessary for a design of good codes, but may not be sufficient. For example, the algorithms used in the programs developed by Ned Seeman (designed for producing sequences suitable for three-dimensional DNA structures), all check for uniqueness of k-length subsequences in the code words (by testing overlapping subsequences of random generated sequences to enforce uniqueness), and none of these properties ensure uniqueness of k-length words [113].

But almost all these theoretical properties are preserved by splicing recombination, and the XPCR method has the convenience that encoding properties need to be studied only for initial sequences. In particular, interesting facts result by extending the concepts of h-comma-freedom and h-infix code as in the following [106].

Definition 3.9. *A language X is strictly h-free if it is h-comma-free and also $X \cap h(X) = \emptyset$. It is said to be h-compliant if, for any words $x, y, u \in A^\star$, $u, xh(u)y \in X$ $(h(u)y \in X,\ xh(u) \in X) \ \Rightarrow\ xy = \lambda$.*

If the initial set of codewords is strictly h-free, then all the sequences that may appear along any splicing computation will not violate the property of h-comma-freedom. Moreover, for any computational system based on splicing one can construct an equivalent one with h-compliance and h-freedom properties, being preserved during any computation. Finally, there exists a method for construction of a strictly h-free (initial) language for a splicing based computation, and it is proved to be close to optimal from the point of view of the efficiency of representing information [106].

In our experiments (see Chapter 6) we adopted a comma-free encoding, following some of the general principles given in [8, 29], by using a program that checked the strings of the pool and primers according to the following strategy. A test $T(n, m)$ is positive when a situation is found such that in a window of n consecutive positions there are at least m discordance positions. This means that when $T(n, m+1)$ is negative then a window there exists with more than $n-m+1$ concordance positions (in this case the program localizes and shows all these win-

dows). Several tests were performed with different values of the parameters n and m related to the size of our primers.

An important *caveat* was the 'primer rotation' phenomenon. When it occurs, a forward primer can play the role of another reverse primer or *viceversa*. We collected the outcomes of many experimental trials that suggested us the values of parameters that ensure a reliable behaviour of the primers. However, oligos and primers we used in the experiment do not have common strings of 5 bases long, apart the expected annealing part, where the concordance is total in a window with the same length of the primer.

The XPCR technique and the extraction algorithm presented in this chapter were introduced by the following publications.

- G. F., C. Giagulli, C. Laudanna, V. Manca, *DNA Extraction by Cross Pairing PCR*, C. Ferretti G. Mauri C. Zandron eds, Preliminary Proceedings of 10th International Meeting on DNA Computing: DNA 10, University of Milano Bicocca, Milan, Italy, June 7-10, pp 193-201, 2004.

- G. F., C. Giagulli, C. Laudanna, V. Manca, *DNA Extraction by XPCR*, C. Ferretti G. Mauri C. Zandron editors, 10th International Workshop on DNA Computing, DNA 10, Milan, Italy, June 2004, Revised Selected Papers, LNCS 3384, Springer, pp 104-112, 2005.

4

On Generating Libraries by XPCR

The art of doing mathematics consists in finding that special case which contains all the germs of generality.

David Hilbert

Although it is now clear that solution of large scale combinatorial search problems, most probably, will be not the future of DNA based computations, new methods, models and techniques are described very often through proposals for a solution to such problems [111]. Namely, a typical DNA recombination is the production of the solution space of instances of SAT problem (see Section 5.1 and, for example, [143, 29, 154]). In fact, combinatorial generation of the solution space of an NP problem is the first step of the Adleman-Lipton extract model in DNA computing [1, 143]. On the other hand, DNA recombination is a very important DNA manipulation, having applications that are beyond the combinatorial mathematical problems. In fact, the production of combinatorial libraries is fundamental to various types of in vitro selection experiments, for selecting new DNA or RNA enzymes (such as ribozymes), or for performing crossover of homologous genes [223] and mutagenesis [150]. It is very important also to sequence genomes and to produce DNA memories.

In DNA computing the classic two methods for initial pool generation were introduced in 1994. The *hybridization-ligation* method was introduced by Adleman in [1] for solving a Hamiltonian Path Problem; it links oligonucleotides hybridized by complementarity and ligase reaction. The other method, called *parallel overlap assembly (POA)* was introduced by Stemmer [223] to perform DNA shuffling, that is, crossover between homologous sequences of genes. It has been then broadly applied in gene construction [105] and gene reconstruction [52]. Its implementation was based on recursive hybridization of oligos and polymerase extension, and it was applied successfully by Kaplan et al. for solving maximal clique problem [122]. Starting from the same encoding schema of Adleman but dealing with single oligonucleotides, the mechanism of POA resembles the Polymerase Chain Reaction in that it extends the oligonucleotides by polymerase, by repeating denaturation, annealing and extension. However, the process is performed without

primers, and the number of target DNA strands does not double every cycle but decreases as the cycle progresses, while the lengths of the DNA strands are increasing. In Figure 4.1 two pictures taken from [108] are reported that clearly depict the two methods. The parallel overlap assembly demands less time and is a better choice in terms of generation speed and material consumption than hybridization-ligation based method [140]; more details about this comparison are given in Section 4.5.

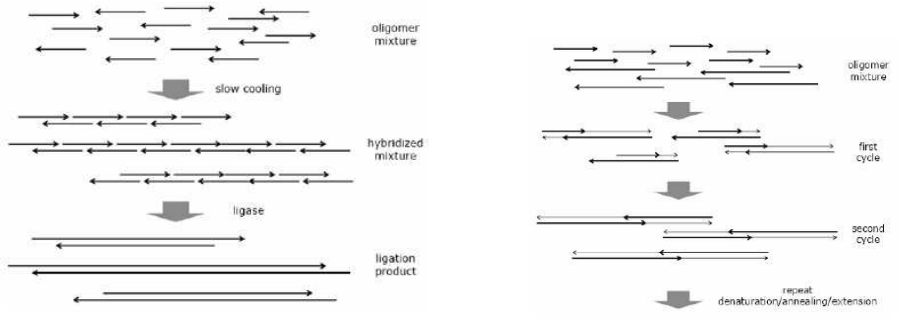


Fig. 4.1. Implementation schema of hybridization/ligation and parallel overlap assembly methods, on the left and right side respectively.

More recently, *mix-and-split* method was introduced in [61] to generate an initial RNA pool for the knight problem, that is a combinatorial library of binary numbers. It was used in [28] to generate an initial pool for a 20-variable 3-SAT problem, the biggest instance solved in a laboratory. It combines a chemical synthesis and an enzymatic extension method, in fact, two half libraries are combined by primer extension while each of them was synthesized chemically on two columns by repetitive mixing and splitting steps. Therefore this method takes the advantages of an extension method but performs a big part of the work by chemical synthesis, which can be quite expensive. Finally, a modified version of PNA-mediated Whiplash PCR was presented in [197], with an in vitro algorithm to evolve approximate solutions of an instance of Hamiltonian Path Problem. It is basically implemented by recursive polymerase extension of DNA hairpins (see Subsection 3.1.1), and performs a programmable crossover operation for generating a combinatorial library for evolutionary solutions, where the main step is realized by restriction-enzyme based methods of crossover.

In Section 4.1 a novel generation method is pursued by means of XPCR; it was successfully tested by the laboratory experiment described in Section 6.4. The implementation approach is similar to the POA, but primers are used to speed up the recombination and the material consumption is minimized. Moreover, for XPCR based algorithm it is proved the existence of some special types of sequences, called *recombination witnesses* (by Definition 4.6), such that when they are present in the final pool, then the pool contains all the solutions of the n-dimensional DNA solution space (see Section 4.3). In Section 4.5 the advantages of XPCR based method with respect to the technologies from literature are pointed out.

Some combinatorial properties of a string generation procedure based on null context splicing rules (implemented by XPCR) are presented in this chapter, as well as a discussion for the generalization of the procedure to the case of non-boolean variables. Firstly, the string recombination via null context splicing rules is presented by the algorithm proposed in Section 4.1, then some combinatorial aspects of it are analyzed in Section 4.2. As a special case, the correctness (as the completeness) of the XPCR-based generation algorithm is obtained by the Proposition 4.2. Whereas nice mathematical results raised from the combinatorial process of generating a discrete space of dimension n via splicing operations are reported in Section 4.4.

4.1 A generation algorithm

The general schema of a recombination procedure consists of an initial pool with a few different kinds of DNA strands, and a sequence of computational steps that finally produces a DNA pool with a variety of different strands that are made of pieces of the initial strands. The generation method here proposed starts from four specific sequences I_1, I_2, I_3, I_4, each constituted by n different strings which can occur in two distinct forms (say X and Y), and, by using only polymerase extension, generates the whole DNA solution space, where all types of possible sequences of the pool are present. The main intuition underlying this quaternary recombination was inspired by [28], where the following sequences were studied to avoid mismatch phenomena.

For the sake of simplicity we describe the method for a solution space of dimension six. From a mathematical point of view, it can be generalized to any dimension n, and to cases where the variables can assume k values with $k > 2$ (see Section 4.4). Let us consider the sequences:

1. Positive: $I_1 = X_1X_2X_3X_4X_5X_6$
2. Negative: $I_2 = Y_1Y_2Y_3Y_4Y_5Y_6$
3. Positive-Negative: $I_3 = X_1Y_2X_3Y_4X_5Y_6$
4. Negative-Positive: $I_4 = Y_1X_2Y_3X_4Y_5X_6$

Starting from these four initial sequences, every element $\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6$, with $\alpha_i \in \{X_i, Y_i\}$ and $i = 1, 2, \ldots, 6$, of the solution space may be generated by means of specific null context splicing rules (by Proposition 4.1). In fact, intuitively, any string $\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6$ can be seen as the concatenation of substrings of I_1, I_2, I_3, I_4 that suitable splicing rules cut and recombine along common substrings.

We start with a pool having only the four initial sequences I_1, I_2, I_3, I_4 that have been extended by a common prefix α and a common suffix β (to perform XPCR procedure).

input the pool $P = \{\alpha I_1\beta,\ \alpha I_2\beta,\ \alpha I_3\beta,\ \alpha I_4\beta\}$
for $i = 2, 3, 4, 5$ **do**
begin

1. **split** P randomly in two new pools P_1 and P_2 (with the same approximate size);

2. **perform** $P_1 := XPCR_{X_i}(P_1)$ **and** $P_2 := XPCR_{Y_i}(P_2)$;
3. **mix** the two pools obtained in the previous step in a pool $P := P_1 \cup P_2$;

end
output the pool P.

If all steps are performed correctly we obtain a pool where all 2^6 combinations $\alpha_1\alpha_2\cdots\alpha_6$ with $\alpha_i \in \{X_i, Y_i\}$ are present (see Proposition 4.2). Moreover, it can be proved that, for any dimension n of the solution space, some sequences exist such that their simultaneous presence in the pool guarantees that all the possible recombinations happened [68]. Four sequences with this property are the *recombination witnesses* (see Definition 4.6 and Proposition 4.8): $X_1X_2Y_3Y_4X_5X_6$, $Y_1Y_2X_3X_4Y_5Y_6$, $Y_1X_2X_3Y_4Y_5X_6$, $X_1Y_2Y_3X_4X_5Y_6$, and in order to verify that all expected recombinations happened in the experiment implementing the XPCR-based procedure, it was enough to check that in the final pool these sequences were present.

4.2 Combinatorial properties

Let us consider n boolean variables $x_1, x_2, \ldots, x_n$, and call X_i and Y_i the values that the i-th variable x_i can assume. We do not use the standard logical symbols T and F for true and false values, because each variable has to be encoded within a different DNA sequence in order to make us able to read the result at the end of the process. Moreover, we are interested to generate a generic binary library beyond the SAT problem.

When n is even, the *initial sequences* of XPCR generation algorithm are the following[1]

1. Positive: $I_1 = X_1X_2X_3X_4\cdots X_{n-1}X_n$
2. Negative: $I_2 = Y_1Y_2Y_3Y_4\cdots Y_{n-1}Y_n$
3. Positive-Negative: $I_3 = X_1Y_2X_3Y_4\cdots X_{n-1}Y_n$
4. Negative-Positive: $I_4 = Y_1X_2Y_3X_4\cdots Y_{n-1}X_n$.

Note that the cases $n = 1$ and $n = 2$ are trivial, because no recombination have to be performed (all combinations are already present as initial sequences), and the corresponding SAT problems do not result interesting, in particular they belong to P class. Thus from now on we assume $n \geq 3$.

We call γ-recombination the operation corresponding to the splicing rule r_γ, which is performed by $XPCR_\gamma$, where the string rewriting rule is

$$\phi\,\gamma\,\psi,\ \ \delta\,\gamma\,\eta \quad \xrightarrow{r_\gamma} \quad \phi\,\gamma\,\eta,\ \ \delta\,\gamma\,\psi$$

[1] otherwise, when n is odd, we have as initial sequences the analogous ones:

1. Positive: $I_1 = X_1X_2X_3X_4\cdots X_{n-1}X_n$
2. Negative: $I_2 = Y_1Y_2Y_3Y_4\cdots Y_{n-1}Y_n$
3. Positive-Negative: $I_3 = X_1Y_2X_3Y_4\cdots Y_{n-1}X_n$
4. Negative-Positive: $I_4 = Y_1X_2Y_3X_4\cdots X_{n-1}Y_n$.

and $\phi, \gamma, \psi, \delta, \eta$ are strings on a predefined alphabet. Of course, a γ-recombination applied to a couple of identical strings works as the identity function, as well as if applied to strings that do not contain the pattern γ.

Given the correctness of the XPCR procedure (from the previous chapter), in the following we identify the application of a splicing rule r_γ (on couples of initial sequences having γ as subsequence) with the running of an $XPCR_\gamma$ (on a pool containing those couples). We observe that, in this context, the order of application of such a kind of rules does not change the product strings.

Let $L = \{\alpha_1\alpha_2 \ldots \alpha_n \mid \alpha_i \in \{X_i, Y_i\},\ i = 1, \ldots, n\}$.

Proposition 4.1. *The n-dimensional solution space L is obtained from the set $A = \{I_1, I_2, I_3, I_4\}$ by sequentially applying all the γ-recombinations with $\gamma \in \{X_2, \ldots, X_{n-1}, Y_2, \ldots, Y_{n-1}\}$.*

Proof. We give a procedure proving that for every recombination $\alpha_1\alpha_2 \ldots \alpha_n$ in L there exists a subset of $\{r_{\alpha_2}, r_{\alpha_3}, \ldots, r_{\alpha_{n-1}}\}$, which rules are sufficient to produce the given recombination starting from the initial strings $\{I_1, I_2, I_3, I_4\}$.

Let us consider the recombination $\alpha_1\alpha_2 \ldots \alpha_n$, and for $i = 2, \ldots, n$, let us call l_i the initial string containing $\alpha_{i-1}\alpha_i$ as a substring (by construction, for each value of i there exists only one of such initial strings). Starting from the initial pool A, the following computation (where φ, ψ and ζ are string variables) generates $\alpha_1\alpha_2 \ldots \alpha_n$ by applying the rules $r_{\alpha_2}, r_{\alpha_3}, \ldots, r_{\alpha_{n-1}}$.

$\varphi := l_2;\ \psi := \lambda;\ \zeta := \lambda;$

for $j = 2, \ldots, n-1$

1. **apply** $\varphi,\ l_{j+1} \xrightarrow{r_{\alpha_j}} \psi,\ \zeta$
2. $\varphi := \psi$

output φ

The length of the output string φ is constant during the computation, it has the same length of the initial strings and of $\alpha_1\alpha_2 \ldots \alpha_n$. The string φ contains the substring $\alpha_1\alpha_2$ before the cycle *for*, and it contains the substring $\alpha_1\alpha_2 \ldots \alpha_i$ after the step corresponding to $j = i-1$. Therefore, after the whole cycle *for* the output string φ is equal to $\alpha_1\alpha_2 \ldots \alpha_n$. □

Since the XPCR-based generation procedure performs all the γ-recombinations with $\gamma \in \{X_2, \ldots, X_{n-1}, Y_2, \ldots, Y_{n-1}\}$ starting from a pool containing the initial sequences, as immediate consequence of the Proposition 4.1 we have the correctness of the XPCR generation algorithm. Note that, in terms of splicing rules, the Proposition 4.1 can be rephrased by saying that the SAT solution space is the null context splicing (or strictly locally testable) language generated by the system $\mathcal{N} = \{\Delta, A, R\}$, where $\Delta = \{a, t, g, c\}$, $A = \{I_1, I_2, I_3, I_4\}$ (with $X_i, Y_i \in \Delta^\star$ for $i = 1, \ldots, n$), and $R = \{r_{X_2}, r_{Y_2}, \ldots, r_{X_{n-1}}, r_{Y_{n-1}}\}$. Another clear consequence is expressed by the following fact.

Corollary 4.2. *Starting from the initial sequences, the XPCR generation algorithm produces all 2^n combinations in time linear with respect to n.*

Proof. By Proposition 4.1 we claim that if all the possible γ-recombinations have happened starting from the initial strands (that is XPCR generation algorithm was entirely executed), then all the 2^n combinations $\alpha_1\alpha_2\ldots\alpha_n$ with $\alpha_i \in \{X_i, Y_i\}$ have been produced in the final pool.

XPCR generation algorithm by definition performs $2(n-2)$ steps by applying all the splicing rules r_γ with $\gamma = X_2, \ldots, X_{n-1}, Y_2, \ldots, Y_{n-1}$, that is, it has a linear complexity with respect the size of the problem n. □

Conversely, for any value of n, the presence of only two specific sequences (recombination witnesses) in the final pool guarantees that all the γ-recombinations, with $\gamma = X_2, \ldots, X_{n-1}$, $Y_2, \ldots, Y_{n-1}$, have been performed starting from a pool containing the four initial sequences. Consequently, by Proposition 4.1, such a presence guarantees that any combination in L is present in the final pool, too.

The existence of recombination witnesses is interesting from both theoretical and experimental points of view, let us explain the theory behind it.

Definition 4.3. *Let $\alpha_i \in \{X_i, Y_i\}$. We say that α_i is positive if it is equal to X_i and is negative if it is equal to Y_i.*

We note that, for $i = 2, \ldots, n-1$ and for any sign (positive or negative) of $\xi_i \in \{X_i, Y_i\}$, a sequence $\alpha_{i-1}\,\xi_i\,\alpha_{i+1}$ has exactly two consecutive variables with a same sign in only two cases:

1. ξ_i has sign equal to the sign of α_{i-1} and different from the sign of α_{i+1} (that is, $\alpha_{i-1}\,\xi_i\,\alpha_{i+1}$ is $X_{i-1}X_iY_{i+1}$ if $\xi_i = X_i$ and is $Y_{i-1}Y_iX_{i+1}$ if $\xi_i = Y_i$),
2. ξ_i has sign equal to the sign of α_{i+1} and different from the sign of α_{i-1} (that is, $\alpha_{i-1}\,\xi_i\,\alpha_{i+1}$ is $Y_{i-1}X_iX_{i+1}$ if $\xi_i = X_i$ and is $X_{i-1}Y_iY_{i+1}$ if $\xi_i = Y_i$).

Definition 4.4. *We call ξ_i-trio factor any substring $\alpha_{i-1}\xi_i\alpha_{i+1}$ where there are exactly two consecutive variables with a same sign, and ξ_i-target a string which includes a ξ_i-trio factor.*

If $i \neq j$ then a ξ_i-target can be a ξ_j-target, and in general, for $i = 2, \ldots, n-1$, the intersections between sets of ξ_i-target strings are nonempty. While a X_i-target sequence cannot be a Y_i-target sequence, for any value of i, because each string of the pool contains either X_i or Y_i.

If we assume that the XPCR based generation algorithm has run on a pool containing the four initial sequences, then the following result holds.

Lemma 4.5. *For any $i = 2, \ldots, n-1$, the rule r_{ξ_i} has been applied iff in the pool there is a ξ_i-target string.*

Proof. Let us consider a fixed value of i.

Only if) We show that the application of the rule r_{ξ_i} on a (uniquely determined) couple of initial strings generates (only) strings including a ξ_i-trio factor.

If i is even, then the application of r_{ξ_i} can be either (case $\xi_i = X_i$)

$$I_1,\ \ I_4 \quad \xrightarrow{r_{X_i}} \quad \phi\, X_{i-1}X_iY_{i+1}\,\eta,\ \ \delta\, Y_{i-1}X_iX_{i+1}\,\psi$$

where ϕ, ψ are respectively a prefix and a suffix of I_1, and δ, η are respectively a prefix and a suffix of I_4, or (case $\xi_i = Y_i$)

$$I_2,\ \ I_3 \quad \xrightarrow{r_{Y_i}} \quad \phi\, Y_{i-1}Y_iX_{i+1}\, \eta,\ \ \delta\, X_{i-1}Y_iY_{i+1}\, \psi$$

where ϕ, ψ are respectively a prefix and a suffix of I_2, and δ, η are respectively a prefix and a suffix of I_3.

While, if i is odd, then we analogously have either (case $\xi_i = X_i$)

$$I_1,\ \ I_3 \quad \xrightarrow{r_{X_i}} \quad \phi\, X_{i-1}X_iY_{i+1}\, \eta,\ \ \delta\, Y_{i-1}X_iX_{i+1}\, \psi$$

where ϕ, ψ are respectively a prefix and a suffix of I_1, and δ, η are respectively a prefix and a suffix of I_3, or (case $\xi_i = Y_i$)

$$I_2,\ \ I_4 \quad \xrightarrow{r_{Y_i}} \quad \phi\, Y_{i-1}Y_iX_{i+1}\, \eta,\ \ \delta\, X_{i-1}Y_iY_{i+1}\, \psi$$

where ϕ, ψ are respectively a prefix and a suffix of I_2, and δ, η are respectively a prefix and a suffix of I_4.

In all cases the ξ_i-recombination r_{ξ_i} generates (only) ξ_i-target strings. Also, if the rule r_{ξ_i} has been applied, then, for each (of two) possible ξ_i-trio factors, in the pool there are ξ_i-target strings containing it.

If) We suppose that the rule r_{ξ_i} was not applied on the pool, and from the following three claims we obtain that no string from the pool includes a ξ_i-trio factor (that is no ξ_i-target is present in the pool).

i) For all $i = 2, \ldots, n-1$, no initial string includes a ξ_i-trio factor. In fact, for any i, the substrings $\alpha_{i-1}\, \xi_i\, \alpha_{i+1}$ of I_1 and I_2 are such that both α_{i-1} and α_{i+1} have sign equal to the sign of ξ_i, and the substrings $\alpha_{i-1}\, \xi_i\, \alpha_{i+1}$ of I_3 and I_4 are such that both α_{i-1} and α_{i+1} have the sign different from the sign of ξ_i.

ii) The rule r_{α_i} such that $\alpha_i \neq \xi_i$ does not produce any ξ_i-target string, because its product strings (as well as its premises) contain α_i and thus do not contain ξ_i.

iii) If a rule r_{α_j} such that $j \neq i$ is applied to strings which do not include a ξ_i-trio factor, then no ξ_i-target is included in the strings produced by the rule. In fact, when a rule r_{α_j} recombines two strings $\phi\alpha_j\psi$ and $\delta\alpha_j\eta$ we have

$$\phi\, \alpha_j\, \psi,\ \ \delta\, \alpha_j\, \eta \quad \xrightarrow{r_{\alpha_j}} \quad \phi\, \alpha_j\, \eta,\ \ \delta\, \alpha_j\, \psi,$$

therefore, any hypothetical ξ_i-trio factor in the product strings should be already present as substring of the premise strings.

In conclusion, ξ_i-target strings are not present in the pool as initial sequences and are not produced by any rule different from r_{ξ_i}. □

4.3 A couple of special strands

Let us now introduce four sequences (including n variable strings) with a special structure:

1. Double positive[2]: $W_1 = X_1X_2\,Y_3Y_4\,X_5X_6\,Y_7 \ldots Z_n$
2. Double negative[3]: $W_2 = Y_1Y_2\,X_3X_4\,Y_5Y_6\,X_7 \ldots Z_n$
3. Positive prefix double negative[4]: $T_1 = X_1\,Y_2Y_3\,X_4X_5\,Y_6Y_7 \ldots Z_n$
4. Negative prefix double positive[5]: $T_2 = Y_1\,X_2X_3\,Y_4Y_5\,X_6X_7 \ldots Z_n$

Definition 4.6. *We call recombination witnesses the strings* W_1, W_2, T_1, T_2.

If one considers any ξ_i-trio factor, then it is a substring of one among the recombination witnesses. Indeed, by construction, these are sequences such that:

1. $X_{i-1}X_iY_{i+1}$ is a subsequence of W_1 if $i \equiv 2 \pmod 4$, of W_2 if $i \equiv 0 \pmod 4$, of T_1 if $i \equiv 1 \pmod 4$, and of T_2 if $i \equiv 3 \pmod 4$.
2. $Y_{i-1}Y_iX_{i+1}$ is a subsequence of W_1 if $i \equiv 0 \pmod 4$, of W_2 if $i \equiv 2 \pmod 4$, of T_1 if $i \equiv 3 \pmod 4$, and of T_2 if $i \equiv 1 \pmod 4$.
3. $Y_{i-1}X_iX_{i+1}$ is a subsequence of W_1 if $i \equiv 1 \pmod 4$, of W_2 if $i \equiv 3 \pmod 4$, of T_1 if $i \equiv 0 \pmod 4$, and of T_2 if $i \equiv 2 \pmod 4$.
4. $X_{i-1}Y_iY_{i+1}$ is a subsequence of W_1 if $i \equiv 3 \pmod 4$, of W_2 if $i \equiv 1 \pmod 4$, T_1 if $i \equiv 2 \pmod 4$, and of T_2 if $i \equiv 0 \pmod 4$.

An intuitive explanation of this property is that the recombination witnesses are periodic with a motif four variables long, thus one finds a same triple of variables by jumping four variables in any of both directions along the string. The motifs are different for each sequence, but each of them is obtained by shifting the signs of another one, and as a consequence we have that for each value of i mod 4 one finds in the four sequences all the four different ξ_i-trio factors.

Definition 4.7. *A set W of strings is a witness set if for every value of ξ_i its strings include a ξ_i-trio factor.*

An example of witness set is $\{W_1, W_2, T_1, T_2\}$. The last results of this section claim that the cardinality of a witness set is greater than or equal to two. At this point it is quite straightforward the following proposition, which makes sense to Definition 4.6.

Proposition 4.8. *If the pool contains a witness set W, then it contains the set L.*

Proof. Since for any ξ_i-trio factor there exists a string in W that is ξ_i-target, then, by Lemma 4.5, the presence of all the elements of W in the pool guarantees that all the ξ_i-recombinations have been performed, with $\xi_i \in \{X_i, Y_i\}$ and $i = 2, \ldots, n-1$. Therefore, by Proposition 4.1, it guarantees that any recombination $\alpha_1\alpha_2 \ldots \alpha_n$ from L is present in the pool. □

A characterization of the recombination witnesses is that they are the only recombinations that cannot be generated from the initial strings by less than $n-2$ splicing rules. In fact, every their substring three variables long is a ξ_i-trio factor for

[2] $Z_n = X_n$ for $n \equiv 1, 2 \pmod 4$, and $Z_n = Y_n$ for $n \equiv 0, 3, \pmod 4$
[3] $Z_n = Y_n$ for $n \equiv 1, 2 \pmod 4$, and $Z_n = X_n$ for $n \equiv 0, 3 \pmod 4$
[4] $Z_n = X_n$ for $n \equiv 0, 1 \pmod 4$, and $Z_n = Y_n$ for $n \equiv 2, 3 \pmod 4$
[5] $Z_n = Y_n$ for $n \equiv 0, 1 \pmod 4$, and $Z_n = X_n$ for $n \equiv 2, 3 \pmod 4$

some ξ_i. By Lemma 4.5, this means that for each (present) recombination witness and for each value of $i = 2, \ldots, n-1$, the splicing rule corresponding to the i-th variable of that sequence must have been applied, that is $n-2$ recombinations have been necessary to generate that recombination witness. Of course, in general, there exist sequences produced by less than $n-2$ splicing rules. For example, the string $X_1X_2Y_3X_4Y_5Y_6$ (for $n = 6$) can be obtained starting from the initial ones by only two rules, in the following way:

1. $I_1,\ I_4 \xrightarrow{r_{X_2}} X_1X_2Y_3X_4Y_5X_6,\ Y_1X_2X_3X_4X_5X_6$
2. $I_2,\ X_1X_2Y_3X_4Y_5X_6 \xrightarrow{r_{Y_5}} Y_1Y_2Y_3Y_4Y_5X_6,\ \mathbf{X_1X_2Y_3X_4Y_5Y_6}$.

Although the four recombination witnesses W_1, W_2, T_1, T_2 are already a good check that all the possible recombinations have been generated by XPCR based generation procedure (see Section 6.4), a stronger result holds. By the following theorem we see that only two of these sequences suffices to guarantee that all the combinations are present in the pool.

Intuitively, it is enough to note that for any $i = 2, \ldots, n-1$, and any sign of ξ_i there are two possible recombination witnesses that are ξ_i-target, while the presence of only one of them suffices to guarantee that the corresponding ξ_i-recombination has been performed. Thus the point is to find a subset of recombination witnesses such that it contains a ξ_i-target for each possible ξ_i.

Theorem 4.9. *If both W_1 and W_2 are present in the pool, then all the strings from L are present, too.*

Proof. Firstly we note that, for any value of $i = 2, \ldots, n-1$, the substrings X_i and Y_i are present within W_1 or W_2. In fact, X_i occurs in W_1 for $i \equiv 1, 2 \pmod 4$ and it occurs in W_2 for $i \equiv 0, 3 \pmod 4$, while Y_i occurring in W_1 for $i \equiv 0, 3 \pmod 4$ and occurring in W_2 for $i \equiv 1, 2 \pmod 4$ and $i \equiv 2 \pmod 4$.

Since for any value of $\xi_i \in \{X_i, Y_i\}$ one of the strings W_1 or W_2 is a ξ_i-target (they include a ξ_i-trio factor by construction), by Lemma 4.5, in a pool containing W_1 and W_2 all the rules r_{ξ_i}, with $\xi_i \in \{X_i, Y_i\}$ and $i = 2, \ldots, n-1$, must have been applied (all the possible γ-recombinations have been performed). Therefore, by Proposition 4.1, all recombinations $\alpha_1\alpha_2 \ldots \alpha_n$ from L are present in the pool. □

In a very similar way the following analogous result is proved.

Corollary 4.10. *If both T_1 and T_2 are present in the pool, then all the strings in L are present, too.*

Proof. For any value of $i = 2, \ldots, n-1$, the variables X_i and Y_i occur as substring of one of the recombination witnesses T_1 and T_2. In fact, for $i \equiv 0, 1 \pmod 4$ the substring X_i occurs in T_1 and for $i \equiv 2, 3 \pmod 4$ it occurs in T_2, while Y_i being a substring of T_1 for $i \equiv 0, 1 \pmod 4$ and of T_2 for $i \equiv 2, 3 \pmod 4$.

Since for any value of i and any value of ξ_i a ξ_i-trio factor can be found in T_1 or T_2, then all the possible ξ_i-target strings are present in the pool. By Lemma 4.5, we claim that all the possible recombinations have been performed, and by Proposition 4.1 we deduce that the pool contains all the elements of L. □

The presence of only a couple of recombinations witnesses suffices to guarantee that the pool contains all the possible recombinations, but the couple has to be $\{W_1, W_2\}$ or $\{T_1, T_2\}$, it cannot contain one between W_1 and W_2 and the other one between T_1 and T_2. In fact, T_1 and W_1 can be present in the pool also when, for example, r_{X_3}, r_{Y_5}, r_{X_7} were not applied (in fact both of them do not contain any X_3-trio factor, Y_5-trio factor, X_7-trio factor), T_1 and W_2 can be similarly present also if r_{X_2} and r_{Y_4} were not performed, and the presence of T_2 and W_1 does not show anything about the application of r_{Y_2}, r_{X_4} or r_{Y_6}. Finally, T_2 and W_2 are present also if, for example, only the recombination r_{X_3} was executed with respect to the third variable (i.e., if r_{Y_3} was not executed).

An obvious characterization of a 'sound' couple of recombination witnesses, that can be $\{W_1, W_2\}$ or $\{T_1, T_2\}$, is that the set of splicing rules necessary to produce its strings cannot be smaller than the set of all the recombination rules r_γ with $\gamma \in \{X_2, \ldots, X_{n-1}, Y_1, \ldots, Y_{n-1}\}$. In fact, for each variable α_i, one string of such a witness set must contain X_i and the other one must contain Y_i.

Therefore, two is the minimum number of recombination witnesses testing the presence of the whole solution space L in the pool.

Corollary 4.11. *The presence of one recombination witness in the pool does not guarantee that all the recombinations belonging to L are present.*

Proof. It is enough to note that in the pool the string W_1 can be present even if all the rules r_{X_i} and r_{Y_j} with $i \equiv 0, 3 \pmod 4$ and $j \equiv 1, 2 \pmod 4$ have been not applied, the string W_2 can be present even if all the rules r_{X_i} and r_{Y_j} with $i \equiv 1, 2 \pmod 4$ and $j \equiv 0, 3 \pmod 4$ have been not applied, the string T_1 can be present even if all the rules r_{X_i} and r_{Y_j} with $i \equiv 2, 3 \pmod 4$ and $j \equiv 0, 1 \pmod 4$ have been not applied, and the string T_4 can be present even if all the rules r_{X_i} and r_{Y_j} with $i \equiv 0, 1 \pmod 4$ and $j \equiv 2, 3 \pmod 4$ have been not applied. □

For the scaling up purpose of DNA computing and for the sake of mathematical generality, we note that all the results here proposed are valid for any value of n. In general, if k (greater than or equal to two) is the number of values that each variable can assume, the minimum number of recombination witnesses testing the presence of the corresponding solution space, as well as the number of initial sequences, depends only on the value of k, as k and k^2 respectively.

4.4 Generalized generation algorithm

If the k different values that the i-th variable can assume are denoted by the elements of $\{Z_i^{(1)}, \ldots, Z_i^{(k)}\}$, for $i = 1, \ldots, n$, then we define the following strings[6].

- $W_1 = Z_1^{(1)} Z_2^{(1)} Z_3^{(2)} Z_4^{(2)} Z_5^{(1)} Z_6^{(1)} Z_7^{(2)} Z_8^{(2)} Z_9^{(1)} \cdots Z_n^{(p_1)}$
- $W_2 = Z_1^{(2)} Z_2^{(2)} Z_3^{(3)} Z_4^{(3)} Z_5^{(2)} Z_6^{(2)} Z_7^{(3)} Z_8^{(3)} Z_9^{(2)} \cdots Z_n^{(p_2)}$

[6] If $n \equiv 1, 2 \pmod 4$ then $p_m = m$ for $m = 1, \ldots, k$. If $n \equiv 0, 4 \pmod 4$ then $p_m = m+1$ for $m = 1, \ldots, k-1$, and $p_k = 1$.

- for $j = 3, \ldots, k-1$
  $W_j = Z_1^{(j)} Z_2^{(j)} Z_3^{(j+1)} Z_4^{(j+1)} Z_5^{(j)} Z_6^{(j)} Z_7^{(j+1)} Z_8^{(j+1)} Z_9^{(j)} \cdots Z_n^{(p_j)}$
- $W_k = Z_1^{(k)} Z_2^{(k)} Z_3^{(1)} Z_4^{(1)} Z_5^{(k)} Z_6^{(k)} Z_7^{(1)} Z_8^{(1)} Z_9^{(k)} \cdots Z_n^{(p_k)}$

Definition 4.12. *We call k-recombination witnesses the strings* $W_1, W_2, \ldots, W_k$.

The analogue of Theorem 4.9 holds also in this general case. Theorem 4.17 is proved after the introduction of Lemma 4.16, that is a generalized version of Lemma 4.5. Let $G_k = \{\alpha_1 \alpha_2 \ldots \alpha_n \mid \alpha_i \in \{Z_i^{(1)}, \ldots, Z_i^{(k)}\}, i = 1, \ldots, n\}$ for $k \geq 2$. It is a generalization of a binary solution space, in fact $G_2 = L$.

All the results of the previous section can be generalized to the general case of $k \in \mathbb{N}$. In fact, the n-dimensional k-solution space G_k formed by all the k^n recombinations of n variables assuming k different values can be generated by applying $k(n-2)$ splicing rules. From an experimental viewpoint, a XPCR-based generation algorithm performing all the XPCR$_\gamma$ with $\gamma \in \{Z_2^{(1)}, \ldots, Z_2^{(k)}, Z_3^{(1)}, \ldots, Z_3^{(k)}, \ldots, Z_{n-1}^{(1)}, \ldots, Z_{n-1}^{(k)}\}$ on a pool containing the following (initial) strings produces the solution space G_k in linear time.

The *initial sequences* $I_1, \ldots, I_{k^2}$ are defined by the expression, for $i, j = 1, \ldots, k$ and n even[7]

$$I_{(i-1)k+j} = Z_1^{(i)} Z_2^{(j)} Z_3^{(i)} Z_4^{(j)} \cdots Z_{n-1}^{(i)} Z_n^{(j)}$$

For an example, let us assume n even and $k = 3$, and suppose that the i-th variable takes values in $\{a_i, b_i, c_i\}$, then the initial sequences are

- (for $i = 1$ and $j = 1, 2, 3$)

$$I_1 = a_1 a_2 \ldots a_n, \quad I_2 = a_1 b_2 a_3 b_4 \ldots b_n, \quad I_3 = a_1 c_2 a_3 c_4 \ldots c_n,$$

- (for $i = 2$ and $j = 1, 2, 3$)

$$I_4 = b_1 a_2 b_3 a_4 \ldots a_n, \quad I_5 = b_1 b_2 \ldots b_n, \quad I_6 = b_1 c_2 b_3 c_4 \ldots c_n,$$

- (for $i = 3$ and $j = 1, 2, 3$)

$$I_7 = c_1 a_2 c_3 a_4 \ldots a_n, \quad I_8 = c_1 b_2 c_3 b_4 \ldots b_n, \quad I_9 = c_1 c_2 \ldots c_n.$$

Note that for any initial sequence $I_{(i-1)k+j}$, i is 'the sign' of the variables with index odd and j is 'the sign' of the variables with index even. The generalized version of the XPCR based generation algorithm follows.

input the pool $P = \{\alpha I_1 \beta, \alpha I_2 \beta, \ldots, \alpha I_{k^2} \beta\}$

for $i = 2, 3, 4, \ldots, n-1$ **do**
begin

1. **split** P randomly in k new pools $P_1, P_2, \ldots, P_k$ (with the same approximate size);

[7] otherwise, when n is odd, we have the analogous ones, for $i, j \in \{1, 2, \ldots, k\}$
$I_{(i-1)k+j} = Z_1^{(i)} Z_2^{(j)} Z_3^{(i)} Z_4^{(j)} \cdots Z_{n-1}^{(j)} Z_n^{(i)}$

2. **perform** $P_j := XPCR_{Z_i^{(j)}}(P_j)$ for $j = 1, \ldots, k$;
3. **mix** the k pools obtained in the previous step in a pool $P := P_1 \cup P_2 \cup \cdots \cup P_k$;

end

output the pool P.

Proposition 4.13. *The n-dimensional k-solution space G_k is obtained from the set $A = \{I_1, I_2, \ldots, I_{k^2}\}$ of initial sequences by applying all the γ-recombinations r_γ with $\gamma \in \{Z_2^{(1)}, \ldots, Z_2^{(k)}, Z_3^{(1)}, \ldots, Z_3^{(k)}, \ldots, Z_{n-1}^{(1)}, \ldots, Z_{n-1}^{(k)}\}$.*

Proof. We give a procedure proving that every recombination $\alpha_1\alpha_2 \ldots \alpha_n$ in G_k is generated by applying the corresponding splicing rules $\{r_{\alpha_2}, r_{\alpha_3}, \ldots, r_{\alpha_{n-1}}\}$ to couples of initial sequences from $\{I_1, I_2, \ldots, I_{k^2}\}$.

Let us consider the recombination $\alpha_1\alpha_2 \ldots \alpha_n$ from G_k, and for $i = 2, \ldots, n$, let us call l_i the initial sequence containing $\alpha_{i-1}\alpha_i$ as substring. By construction, for each value of i there exists only one of such an initial string. In fact, if α_i is equal to Z_i^a and α_{i-1} is equal to $Z_{i-1}^{(b)}$, then $\alpha_{i-1}\alpha_i$ is a subsequence of $I_{(b-1)k+a}$ if $i \equiv 0 \pmod 2$, and it is a subsequence of $I_{(a-1)k+b}$ if $i \equiv 1 \pmod 2$.

The following computation (where φ, ψ and ζ are string variables) generates $\alpha_1\alpha_2 \ldots \alpha_n$ by applying the recombinations $r_{\alpha_2}, r_{\alpha_3}, \ldots, r_{\alpha_{n-1}}$ on some initial sequences from A.

$\varphi := l_2$; $\psi := \lambda$; $\zeta := \lambda$;

for $j = 2, \ldots, n-1$

1. **apply** $\varphi,\ l_{j+1} \xrightarrow{r_{\alpha_j}} \psi,\ \zeta$
2. $\varphi := \psi$

output φ

The length of the output string φ is constant during the computation, it has the same length of the initial strings and of $\alpha_1\alpha_2 \ldots \alpha_n$. The string φ contains the substring $\alpha_1\alpha_2$ before the cycle *for*, and it contains the substring $\alpha_1\alpha_2 \ldots \alpha_i$ after the step corresponding to $j = i-1$. Therefore, after the whole cycle *for* the output string φ is equal to $\alpha_1\alpha_2 \ldots \alpha_n$. □

Note that any recombination is generated by (at most) $n-2$ splicing rules, regardless the number of values that each variable can assume. The generalization of Lemma 4.5 follows.

Definition 4.14. *Let $\alpha_i^{(u)}, \xi_i^{(v)} \in \{Z_i^{(1)}, \ldots, Z_i^{(k)}\}$. We say that $\alpha_i^{(u)}$ and $\xi_j^{(v)}$ have the same sign if $u = v$, otherwise we say they have a different sign.*

Definition 4.15. *We call $\xi_i^{(h)}$-trio factor any substring $\alpha_{i-1}^{(u)}\,\xi_i^{(h)}\,\alpha_{i+1}^{(v)}$ where there are exactly two consecutive variables with a same sign, and $\xi_i^{(h)}$-target a string which includes a $\xi_i^{(h)}$-trio factor.*

Let us assume that the XPCR based generation algorithm has run on a pool containing the k initial sequences, then

Lemma 4.16. *Any rule $r_{\xi_i^{(h)}}$ has been applied iff in the pool there is a $\xi_i^{(h)}$-target string.*

Proof. Let us consider a fixed value of i.

Only if) We show that the application of the rule $r_{\xi_i^{(h)}}$ on a couple of initial sequences generates $\xi_i^{(h)}$-target strings. Such a couple is not uniquely determined, in fact for each value of $\xi_i^{(h)}$ there are k different initial sequences that contain $\xi_i^{(h)}$ as substring.

We can apply the rule $r_{\xi_i^{(h)}}$ on initial sequences having $\alpha_{i-1}^{(u)}\xi_i^{(h)}\alpha_{i+1}^{(u)}$ as subsequence, where either $u = h$ or $u \neq h$. Since there is only the initial sequence $I_{(h-1)k+h}$ having (all) adjacent variables with the same sign h, then we distinguish only two kinds of couples (of initial sequences) by means of the sign of the variables adjacent to $\xi_i^{(h)}$: or the couple contains the sequence $I_{(h-1)k+h}$ or it does not.

Let us assume that i is even. If the couple of premises contains the sequence $I_{(h-1)k+h}$, then the application of $r_{\xi_i^{(h)}}$ works as in the following:

$$I_{(h-1)k+h},\ \ I_{(u-1)k+h} \quad \xrightarrow{r_{\xi_i^{(h)}}} \quad \phi\,\alpha_{i-1}^{(h)}\xi_i^{(h)}\alpha_{i+1}^{(u)}\,\eta,\ \ \delta\,\alpha_{i-1}^{(u)}\xi_i^{(h)}\alpha_{i+1}^{(h)}\,\psi$$

where ϕ, ψ are respectively a prefix and a suffix of $I_{(h-1)k+h}$, and δ, η are respectively a prefix and a suffix of $I_{(u-1)k+h}$. In the non-trivial case that $u \neq h$ (that is all the times that the rule produces a new recombination), then $\xi_i^{(h)}$-target strings are produced.

If the couple of premises does not contain the sequence $I_{(h-1)k+h}$ (case $u, v \neq h$), then the application of $r_{\xi_i^{(h)}}$ works as in the following:

$$I_{(v-1)k+h},\ \ I_{(u-1)k+h} \quad \xrightarrow{r_{\xi_i^{(h)}}} \quad \phi\,\alpha_{i-1}^{(v)}\xi_i^{(h)}\alpha_{i+1}^{(u)}\,\eta,\ \ \delta\,\alpha_{i-1}^{(u)}\xi_i^{(h)}\alpha_{i+1}^{(v)}\,\psi$$

where ϕ, ψ are respectively a prefix and a suffix of $I_{(v-1)k+h}$, and δ, η are respectively a prefix and a suffix of $I_{(u-1)k+h}$. Note that in this case we do not have production of $\xi_i^{(h)}$-target strings.

Let us assume that i is odd. If $I_{(h-1)k+h}$ is a premise, then the rule $r_{\xi_i^{(h)}}$ works as in the following, by producing $\xi_i^{(h)}$-target strings if $u \neq h$:

$$I_{(h-1)k+h},\ \ I_{(u-1)k+h} \quad \xrightarrow{r_{\xi_i^{(h)}}} \quad \phi\,\alpha_{i-1}^{(h)}\xi_i^{(h)}\alpha_{i+1}^{(u)}\,\eta,\ \ \delta\,\alpha_{i-1}^{(u)}\xi_i^{(h)}\alpha_{i+1}^{(h)}\,\psi$$

where ϕ, ψ are respectively a prefix and a suffix of $I_{(h-1)k+h}$, and δ, η are respectively a prefix and a suffix of $I_{(u-1)k+h}$.

Otherwise (case $u, v \neq h$), we have:

$$I_{(h-1)k+v},\ \ I_{(h-1)k+u} \quad \xrightarrow{r_{\xi_i^{(h)}}} \quad \phi\,\alpha_{i-1}^{(v)}\xi_i^{(h)}\alpha_{i+1}^{(u)}\,\eta,\ \ \delta\,\alpha_{i-1}^{(u)}\xi_i^{(h)}\alpha_{i+1}^{(v)}\,\psi$$

where ϕ, ψ are respectively a prefix and a suffix of $I_{(h-1)k+v}$, and δ, η are respectively a prefix and a suffix of $I_{(h-1)k+u}$. Note that in this case we do not have production of any $\xi_i^{(h)}$-target string.

In conclusion, starting from the sequence $I_{(h-1)k+h}$ and any other initial sequence containing $\xi_i^{(h)}$, the $\xi_i^{(h)}$-recombination generates $\xi_i^{(h)}$-target strings. Also, if the rule $r_{\xi_i^{(h)}}$ has been applied, then for each (of the $k-1$) possible $\xi_i^{(h)}$-trio factor, in the pool there are $\xi_i^{(h)}$-target strings containing it.

If) We suppose that the rule $r_{\xi_i^{(h)}}$ was not applied on the pool, and from the following three claims we obtain that no string from the pool includes a $\xi_i^{(h)}$-trio factor (that is, no $\xi_i^{(h)}$-target is present in the pool).

i) By construction, for every value of i ($i = 2, \ldots, n-1$) and for every sign h ($h = 1, 2, \ldots, k$), no initial string includes a $\xi_i^{(h)}$-trio factor. In fact, for any i, and in each of the k initial sequences containing a $\xi_i^{(h)}$, the substring $\alpha_{i-1}\ \xi_i^{(h)}\ \alpha_{i+1}$ has both the variables α_{i-1} and α_{i+1} having a same sign u (which is equal to h in the sequence $I_{(h-1)k+h}$ and it is different from h in all the other $k-1$ initial sequences), thus it is not a $\xi_i^{(h)}$-trio factor. By definition, we may conclude that no initial sequence is a $\xi_i^{(h)}$-target string.

ii) The rule $r_{\xi_i^{(v)}}$ such that $v \neq h$ does not produce any $\xi_i^{(h)}$-target string, because its product strings (as well as its premises) contain $\xi_i^{(v)}$ and thus do not contain $\xi_i^{(h)}$.

iii) If a rule r_{α_j} such that $j \neq i$ is applied to strings which do not include a $\xi_i^{(h)}$-trio factor (for any value of h), then no $\xi_i^{(h)}$-target strings are produced by the rule. In fact, when a rule r_{α_j} recombines two strings $\phi\alpha_j\psi$ and $\delta\alpha_j\eta$ we have

$$\phi\ \alpha_j\ \psi,\ \ \delta\ \alpha_j\ \eta \quad \xrightarrow{r_{\alpha_j}} \quad \phi\ \alpha_j\ \eta,\ \ \delta\ \alpha_j\ \psi$$

therefore, any hypothetical ξ_i-trio factor in the product strings should be already present as substring of the premise strings.

In conclusion, $\xi_i^{(h)}$-target strings are not present in the pool as initial sequences and are not produced by any rule different from $r_{\xi_i^{(h)}}$. □

From the above (first part of the) proof we deduce an interesting difference with respect to the case $k = 2$. That is, when $k > 2$ and starting from the initial sequences, the rules r_γ generate a set of strings which properly contains the γ-target ones. This observation opens the question whether less than k recombination witnesses can be made, with a different structure, in such a way that they test the presence of the whole solution space. In fact, for $k > 2$ it seems that also the presence of a sequence containing a $\alpha_{i-1}^{(u)}\ \alpha_i^{(h)}\ \alpha_{i+1}^{(v)}$, while u, v, and h being three different values from the set $\{1, 2, \ldots, k\}$, could guarantee that $r_{\alpha_i}^{(h)}$ has been applied.

However, we conclude this section by proving one of the main results introduced: the presence of k particular sequences in the final pool guarantees that the

whole n-dimensional k-solution space G_k was generated by $k(n-2)$ null context splicing rules starting from the k^2 initial sequences.

Theorem 4.17. *If the k-recombination witnesses $W_1, W_2, \ldots, W_k$ are present in the pool, then all the k^n strings of G_k are present, too.*

Proof. For any possible value of $Z_i^{(h)}$ we may find one k-recombination witness such that $Z_i^{(h)}$ occurs as substring. In particular, it occurs as substring of W_h for $i \equiv 1, 2 \pmod 4$ and as substring of W_{h-1} (or W_k if $h = 1$) when $i \equiv 0, 3 \pmod 4$.

For $i = 2, \ldots, n-2$, when $Z_i^{(h)}$ is a substring of a k-recombination witness, the corresponding substring $Z_{i-1}^{(u)} Z_i^{(h)} Z_{i+1}^{(v)}$ is a $Z_i^{(h)}$-trio factor (indeed one value v or u is equal to h and the other one is different). In other words, for any value of $Z_i^{(h)}$ there exists a $Z_i^{(h)}$-target string among the k-recombination witnesses.

Therefore, by Lemma 4.16, in a pool containing all the k-recombination witnesses all the splicing rules required by the algorithm have been applied. By Proposition 4.13, this fact guarantees that the whole k-solution space G_k is present in the pool. □

Unless to change the structure of the recombination witnesses, they all need to be present simultaneously in the pool to guarantee the presence of the whole solution space.

Corollary 4.18. *The presence of k-1 k-recombination witnesses in the pool does not guarantee that all the recombinations of G_k are present.*

Proof. Let $j \in \{1, \ldots, k\}$ and $i, h \in \mathbb{N}$ such that $i \equiv 1, 2 \pmod 4$ and $h \equiv 0, 3 \pmod 4$. By construction, none of the $k-1$ recombination witnesses $W_1, \ldots, W_{j-1}, W_{j+1}, \ldots, W_k$ is a $Z_i^{(j)}$-target string (and neither a $Z_h^{(j+1)}$-target string) because they do not contain $Z_i^{(j)}$ (or $Z_h^{(j+1)}$). Therefore, by Lemma 4.16, $W_1, \ldots, W_{j-1}, W_{j+1}, \ldots, W_k$ might be present in a pool where all the splicing rules but $r_{Z_i^{(j)}}$ and $r_{Z_h^{(j+1)}}$ were performed, and where, for example, the recombinations of G_k containing $Z_i^{(j)}$ are not all present. □

A future research on the subject of this section could be to study the case (and to find the minimum number of recombination witnesses) where different variables may assume a different number of values.

4.5 Advantages of XPCR generation

Apart the mathematical speculations, XPCR-based generation algorithm is a novel method to generate DNA libraries, that has several advantages with respect to the techniques from the DNA computing literature. POA and hybridization/ligation methods (that are described in the introduction of this chapter) were reviewed in [140] as initial pool generation methods of combinatorial problems. From a theoretical comparison and a few experimental simulations, the conclusion was suggested that POA is a better choice in terms of generation speed, material

consumption and efficiency [108]. In the following, we both recall the advantages of POA with respect to the hybridization/ligation methods and compare our method with POA.

In the light of scalability, one may observe that POA maintains the population size (i.e. the number of DNA molecules) throughout the procedure, whereas, in the hybridization/ligation method, the population size decreases as reaction progresses (thus, as the problem size increase, the required initial pool size increases dramatically). In fact, initially two single stranded DNA molecules partially hybridize in the annealing step and then they are extended by polymerase. The elongated DNA molecules are denatured to two single-stranded DNA in the next denaturation step, and they are subjected to the annealing reaction at the next cycle. Therefore, POA does maintain the population size and the population size can be decided by varying the initial number of oligos.

Hence, initial pool generation by POA requires fewer strands than hybridization/ligation method to obtain similar amount of initial pool DNA molecules, because complementary strands are automatically extended by polymerase. This is an enormous advantage, because of DNA molecules synthesis cost, probability of implementation errors, and physical limits about the DNA quantity contained by a test tube [140]. However, the initial amount of DNA molecules of POA is comparable with that one of the library itself. Instead, XPCR generation method has a constant initial amount of DNA molecules, that is four sequences n variables long, where n is the size of a problem with Boolean variables. From this point of view, XPCR based generation method results more economic, efficient and scalable than POA.

Moreover, both POA and XPCR methods are cheaper and more rapid than hybridization/ligation system, because they do not require phosphorylation of oligos, which is prerequisite for the ligation of oligos (modified oligos for ligation takes up most of the expenses), and they demand less time than hybridization/ligation method. Hybridization requires one hour and an half, while ligation requires more than 12 hours. POA for 34 cycles requires two hours, XPCR-based generation method in $n-2$ the time of a XPCR (that is approximately an hour) generates the whole library. After 30 cycles of POA process, various sizes of double-stranded DNA strands are generated. The average extension length continuously increased with each cycle though, which is because extended DNA strands are used as templates for longer strand generation, and the ratio of DNA strands which is long enough to contain optimal solution, is very low [140]. This is mainly due to the rapid decrease of the extension efficiency. If we define the extension efficiency of each cycle as the ratio of annealing event which forms extendable duplex to total annealing event, then the extension efficiency drop dramatically after only few POA cycles. The reason is that non-extendable annealing event increases with cycles, while non-extendable duplex formation (with the $3'$ part of the two single stranded molecules dangling) rapidly increasing with each step.

The main improvement of XPCR generation method with respect to POA is the lack of material waste. No molecules with different lengths are expected to be produced beside to the amplifications of XPCR, which amplifies the recombined strings while maintaining a constant amount of the pieces that are recombined (see Section 3.3). A practical advantage is that the XPCR generation process can

be monitored in each elementary step by an electrophoresis to eliminate eventual unspecific products, because the valid lengths are all known in every step. Namely in the first step of (the for cycle of) the generation algorithm, if one starts with strings long 100bp and the second variable is the substring from the 20th position to the 40th one, then the XPCR process amplifies strings long 40bp and 80bp in the first (cutting) stage and 100bp in the last (recombination) one. Thus an electrophoresis can be performed in any step both to clean the signal and to scan the whole process, so that the method can be correctly iterated as many times as required by the library size.

In the case of POA instead, the valid strands ratio decreases with each cycle because the DNA strands which extended by mis-hybridization are not eliminated or recovered during the process. Moreover, as the DNA strands are elongated, the possibility of a unspecific annealing increases. These are a few reasons reported in [140] to explain why the invalid strands generated by mis-hybridization accumulate during POA cycles, that is why the valid strand ratio of longer strands are lower than that of shorter strands. The main problem is thus the correctness of the POA method as the size of the library increases, since available material to generate new recombinations decreases with cycles. Unfortunately, "POA was not as efficient as we expected" and "the efficiency of POA is not enough to be applied to initial pool generation", J. Y. Lee, H-W. Lim, S-I. Yoo, B-T. Zhang, T. H. Park, [140]. Apart the efficiency, a critical point for a successful POA is the sequence design for initial strands, especially the specificity of $3'$-end. Since template strands behave as primers, the elaboration of $3'$-end sequence design must be considered carefully to prevent an extension after unspecific annealing.

As a conclusion of this comparison between XPCR and POA methods, we point out that the idea on which their implementations are based is similar: we start with few strings, and by means of annealing and polymerase extension we generate all the other ones. But the strategy and the efficiency of XPCR based generation method is much more promising, although more experiments should be carried out to test the expected efficiency for bigger dimension of the problem, and although the primer and sequence design needs to be appropriate to avoid amplification of false positives. For XPCR, the use of primers (which are present in excess in the pool) directs the process by speeding up the generation and by avoiding material consumption, thus no unspecific or redundant material is expected.

A comparison between the implementation efficiency of XPCR generation algorithm and mix and split method, the most recent and efficient technique used so far [61, 28], could not be as detailed as above, because the two procedures follow different approaches. Though we would like to briefly point out some similarities and differences. Both methods require a same number of steps and a same algorithmic strategy[8], but the scalable implementation of XPCR based algorithm promises to be *i)* cheaper, as the first stage of the mix and split method is performed by chemical synthesis of half of the library, *ii)* not less efficient, as in both cases the algorithm is based on enzymatic primer extension, and *iii)* more user friendly (easy to implement), as only PCR and electrophoresis techniques have to be performed.

[8] An anonymous referee from the eleventh international meeting on DNA computing [77] called our method "mix and split PCR".

On the other hand, the existence of the recombination witnesses (see Section 4.3) is a check on the experimental *iter* of our method that represents an actual advantage over all the other existing methods ("The concept of requiring only two witnesses for the existence of a complete library is a novel and interesting one", anonymous referee[8]). They provide us with a control of any eventual experimental error which usually cannot be excluded in any DNA procedure. The idea here is to give the generation method together with a check (couple of recombination witnesses) to control that the method has been implemented correctly, anytime one executes it.

Of course, from a theoretical point of view, we know that the whole library is generated because of the correctness of XPCR procedure and XPCR based generation algorithm (i.e., the null context splicing rules required by the algorithm and implemented by XPCR are enough to generate the expected library). But the operator could possibly wrong or skip some step, or some PCR could not work well, and unexpected experimental pitfalls could alter the result. If the algorithm would be automated we would have even less possibilities to check any step with electrophoresis, and we should trust just the theoretical correctness, as the other methods usually do. In general, the only way to verify that a library has been entirely generated is to check each presence experimentally, with an exponential number of steps. Instead, to test the completeness of a library generated by a quaternary recombination of strings via application of null context splicing rules, it is enough to check the presence of two (recombination witnesses) strings. As we have seen in the previous sections, this result came up from combinatorial speculations, while from a biological point of view, we may assume that the XPCR step, wide experimentally tested, works in parallel and correctly, by involving all the expected molecules.

However, as a kind of limit of all methods based on PCR, we stress the sensitivity of such a reaction: it may be hard to control a process, because unwanted amplification may overcome quite easily, and for any kind of error that should occur, it is amplified with the same rate of the solution. On the other hand, in those cases where these processes work, they work very well, fast and efficiently. Besides, polymerase chain reaction just resembles in vitro the process of autoduplication of DNA, thus it is strictly linked to the way cells replicate their genetic code. Also, the aberrant proliferation of a cell very often characterizes its status of tumored cell, thus in a sense it could be particularly important and interesting to learn to control processes involving DNA replication.

The generation algorithm presented in this chapter, as well as its generalized form and the combinatorial results, were (respectively) introduced in the papers:

- G. F., V. Manca, C. Giagulli, C. Laudanna, *DNA Recombination by XPCR*, Revised Selected Paper in A. Carbone, N.A. Pierce Eds: DNA11, June 6-9 2005, London, Ontario, Canada, LNCS 3892, pp 55-66, 2006.

- G. F., *Combinatorial Properties of DNA Libraries Generated via Null Context Splicing Rules*, Discrete Applied Mathematics, submitted.

5

Applications of XPCR

Nature is an infinite sphere of which the center is everywhere and the circumference nowhere.

Blaise Pascal[1]

The hope of effectively solving NP-problems seems no longer pursued by current trends of DNA computing. Indeed, after the result proposed in [28], where a 20 variable instance of SAT was solved in laboratory, researchers have apparently abandoned this line of research, and to experimentally progress beyond this result seems unlikely. However, the exploration of methods and technologies that are interesting in itself or useful for something other than solving NP-complete problems receives wide interest. Although several applications of such procedures regard genetics and medicine (see Section 5.5), very often NP-complete problems represent a nice way to show that the method or technology works, and several algorithms solving combinatorial problems are suggested. This point is faced in further detail in Chapter 1, along with related references.

In the last two chapters we have presented a method of extraction and one of generation respectively, both implemented on the basis of the XPCR procedure. In this chapter we present more applications of the XPCR technique, such as a theoretical improvement (with a lower complexity) of the XPCR generation method in Subsection 5.3.1, and a mutagenesis algorithm (in Section 5.4) that was tested in laboratory by the experiment reported in Section 6.5.

After a presentation of the SAT problem and of its intrinsic complexity in Section 5.1, a polymerase based algorithm for solving an instance of SAT with significant size is proposed in Section 5.3. Its extraction step (Subsection 5.3.2) should be implemented by a tricky combination of enzymatic operations, such as polymerase extension and endonuclease cutting. The aim of Section 5.2 is to briefly explain the operations performed by restriction enzymes, since the use of a specific endonuclease is required in the extraction step of the algorithm. Some fascinating applications of DNA biotechnologies are outlined in a last section.

[1] *Pensées*, 1670. Quoted in E Maor, *To infinity and beyond*, Princeton, 1991.

5.1 The satisfiability problem (SAT)

One of the most intriguing problem to approach by DNA computations is SAT, which has a very simple definition. Given a propositional formula formed by n boolean variables connected by the logical connectives *or*, *and* and *not*, the SAT problem consists in deciding the existence of an assignment of true values to the variables such that it SATisfies the formula. Every instance of a SAT problem can be written in conjunctive normal form, that is as the conjunction of equations formed by disjunctions and negations on the variables. A k-$SAT(n, m)$ problem, in particular, may be seen as a boolean system of m disjunctive equations (clauses) on n variables $x_1, \ldots, x_n$ and their negations $\neg x_1, \ldots, \neg x_n$ (that are all called literals l_i, where $i = 1, \ldots, n$ and $l_i \in \{x_i, \neg x_i\}$), where each equation contains at most k literals. If there exists an assignment satisfying the given formula we say that the formula is satisfiable, and it belongs to the SAT class, otherwise we say that it is unsatisfiable.

For example, the 3-$SAT(2, 2)$ toy instance $(x \vee y) \wedge (x' \vee y')$, where x' is the negation of x and y' is the negation of y, was the first to be solved with DNA computations, in [143]. There have been huge developments from that experiment (see for example [94,61,154]), and instances of SAT problem up to twenty variables were solved by encoding data in linear duplex DNA molecules [28]. Recent successful attempts for solving SAT were done by linear assembly of DNA tiles [237] and by graph self-assembly, where the problem is encoded in duplex and branched junction DNA molecules such that the graph can self assemble if and only if there is a solution to the problem [118].

If $k = 1, 2$ then a k-SAT problem is solvable with a deterministic Turing machine in polynomial time, i.e., k-$SAT \in P$. The case $k = 1$ is trivial, while an algorithm of computational complexity $O(n + m)$ for the 2-$SAT(n, m)$ problem is well known [10]. The case $k \geq 3$ is more interesting, because then k-$SAT \in NP$. SAT was the first problem shown to be NP-complete (Cook, '71), and it is nowadays one of most widely studied NP-complete problems. A remarkable consequence of the theory of NP-completeness is that a large collection of problems (several thousand) from many different fields can be represented as instances of the k-SAT problem. If $k \geq 3$ then any instance of k-SAT can be reduced to an instance of 3-SAT at the cost of linearly increasing the size of the problem, therefore in the following we will consider only the 3-SAT problem, in order to facilitate the explanation.

The fastest known heuristic to solve 3-SAT is the Davis-Putnam procedure [90] that is complete and 'often' computes in linear time; it performs a backtrack search through the space of possible truth assignments. Although NP-complete problems are believed to require exponential time to solve in the worst case, the typical-case behaviour is difficult to characterize. Yet, such typical-case properties are most relevant in practical applications.

In terms of what is known theoretically about the probability of satisfiability for random 3-SAT, the *threshold conjecture* [132] is that there is some specific ratio of clauses to variables above which the probability of satisfiability approaches 1, and below which it approaches 0. The intuitive explanation for this phenomenon is that at low values of such a ratio we have relatively few clauses and many variables,

which means that the formula is relatively easy to satisfy (the problem is under-constrained and a satisfying assignment can be 'guessed' quickly). At high values of this ratio, the problem is over-constrained and generally no satisfying assignment exists. The most interesting formulas can be found at the transition from the under-constrained to the over-constrained area.

In general, NP-complete problems exhibit 'phase transition' phenomena, analogous to those in physical systems, while the hardest problems occurring at the phase boundary. Across this phase dramatic changes occur in the computational difficulty, and the problems become easier to solve away from the boundary [217]. A more accurate analysis about the relationship between the computational complexity of SAT problems and the phase transition occurring in some physical models as spin glasses [168] was developed in [70], where the application of two methods of the statistical mechanics (the annealed estimate and the replica method) to study k-SAT is discussed.

However, for a 3-$SAT(n, m)$, experimental evidence strongly suggests a threshold with $\frac{m}{n} \approx 4.2$. In Section 5.3 we consider an algorithm to solve a significant instance of a random 30-variable 3-SAT problem with 126 clauses, where one has the variables $x_1, x_2, \ldots, x_{30}$, and the clauses $C_1, C_2, \ldots, C_{126}$, that are the disjunction of at most three literals.

The crossover point divides the space of satisfiability problems into three regions: the under-constrained region below the crossover point, the critically-constrained region in the neighborhood of the crossover point, and the over-constrained region above the crossover point. Each of these regions is interesting for different reasons. Generally the commercially important satisfiability and constraint-satisfaction problems are optimization problems: one wants to minimize costs subject to a given set of constraints. Setting the cost threshold too high results in an under-constrained problem. If instead the cost is set just right one obtains a critically-constrained problem. Similarly for over-constrained problems. If one has an optimization problem to solve, and one does not have sufficiently powerful algorithms to solve it in the critically-constrained region (which is usually the case for realistically complex problems), then the only choice is to loose the cost threshold and to move the problem into the under-constrained region. Thus the under-constrained region is important because in practice it is where optimization problems are usually 'solved'. Clearly the critically-constrained region is important because it is where we must work if we have to solve optimization problems exactly. Finally, the over-constrained region is important because showing over-constrained problems to be unsolvable corresponds to showing optimality of solutions to optimization problems.

The location of the crossover point is of both theoretical and practical relevance. It is theoretically interesting since, a threshold exists for any value of k, but determining the exact location of the threshold remains a difficult open problem, although there has been substantial progress in narrowing the theoretical bounds on the transition, and in the case of 3-SAT, the number of constraints required for crossover appears to be almost (but not exactly) a linear function of the number of variables in the problem. One reason for the limitations of analytic results is the complexity of the analysis required. Another is that they are asymptotic with respect to the number of variables, so they do not necessarily tell us much about

formulas that have only a modest number of variables (say, up to a few thousand) as encountered in practice. On the other hand, the location of the crossover point is of practical interest since empirically the hardest problems seem to be found near that area, thus it makes sense to test candidate algorithms on these hard problems. This means that if one knows the location of the crossover point, then for random problems (i.e., problems with no structure) the number of clauses can be used as a predictor of satisfiability.

In order to solve a SAT instance with 30 variables and 126 clauses we are going to present a DNA algorithm based simply on enzymatic operations. So far, along this thesis, we have seen the extension action of polymerase enzymes (see Section 3.1); another class of enzymes very important in nature are the restriction enzymes, that usually cut molecules.

5.2 Restriction enzymes

The restriction enzymes are known as molecular scalpels because of their cut precision, and they perform DNA computations with an efficiency of 100%. They recognizes specific sequences of nucleotides, called sites, in a double stranded DNA molecule and cut the molecule by destroying the phosphodiester bonds at those specific places of the two strands. Some enzymes perform a *blunt* cut, that is straight both strands (no overhang), whereas others leave dangling single-stranded portions (see Figure 5.1 from [69]), called overhangs.

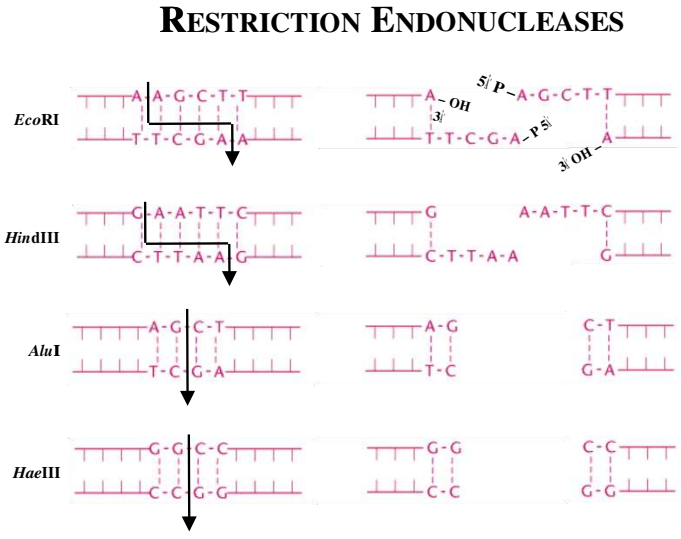


Fig. 5.1. Endonucleases action.

Endonucleases come from different organisms and their names are given according to their origin. There are about one hundred known and commercially available restriction enzymes. In Section 5.3 the algorithm proposed requires just one of these enzymes, called *SmaI*; it is a restriction endonuclease having the restriction site CCCGGG. SmaI bluntly cuts (only) double stranded molecules, by separating CCC and GGG without any overhang, as in Figure 5.2 taken from [71].

It means that, when we put this enzyme in a pool P, all the double strands containing CCCGGG are cut between the last C and the first G. We refer to $\mathbf{SmaI_{cut}(P)}$ as the pool containing the products of SmaI acting on P.

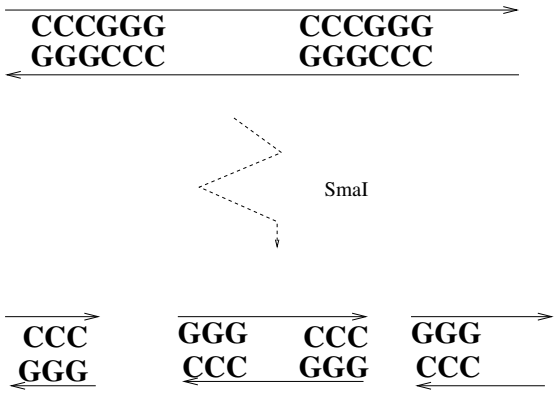


Fig. 5.2. Cut operation of enzyme SmaI.

Different enzymes recognize different sites, and even if they have the same site, they may cut the molecule in a different way. For example, the enzyme *XmaI* has the same site of *SmaI*, but after cutting, XmaI leaves as $5'$ overhang CCGG.

5.3 An enzyme-based algorithm to solve SAT

In this section we present a DNA algorithm for solving SAT based on polymerase extension, enzymatic cuts, and electrophoresis, that are considered standard, cheap, easy to implement, and efficient methods to perform DNA computations (whose notations are given in Chapter 3). It follows the generate-and-search sche'ma of the classical extract model suggested by Adleman and Lipton [1, 143].

The generation step of this algorithm is implemented by a optimized application of the XPCR. In fact, the XPCR generation algorithm proposed in [77] and explained in the previous chapter starts from *four* specific DNA sequences and generates the whole SAT solution space of 2^n different sequences in linear time, instead the generation procedure considered for the following algorithm starts from *two* specific DNA sequences and performs $\lceil \log_2 n \rceil$ recombinations. The way it works is detailed in Subsection 5.3.1.

The extraction step is implemented by a suitable combination of enzymatic operations. In particular it needs a polymerase extension and 'blunt' cuts of a

restriction enzyme, and for example we choose to use SmaI endonuclease. The idea of this extracting step was inspired by [209], where hairpin formations of single strands were used as test of inconsistent assignments of variables and destroyed by a restriction enzyme. However, a really different algorithmic strategy is used, where polymerase extension is an important feature to select the 'good' strings. Also, we manipulate a solution space with 2^n elements instead of one with 3^m elements, where m is the number of clauses of the considered SAT instance.

We outline the algorithm for solving a random instance of a 3-$SAT(30, 126)$ problem. We encode the variables with the DNA sequences $X_1, X_2, \ldots, X_{30}$ respectively, and their negations $\neg x_1, \neg x_2, \ldots, \neg x_{30}$ with the sequences $Y_1, Y_2, \ldots, Y_{30}$ respectively. For the sake of simplicity, we say that the literal l_i is encoded by the DNA sequence L_i, meaning that, if $l_i = x_i$ then $L_i = X_i$, and if $l_i = \neg x_i$ then $L_i = Y_i$. We consider a sequence γ containing the restriction site of SmaI[2], and the following two initial sequences:

$$Z_1 = \alpha X_1 \gamma X_2 \gamma X_3 \cdots \gamma X_{30} \beta \quad \text{and} \quad Z_2 = \alpha Y_1 \gamma Y_2 \gamma Y_3 \cdots \gamma Y_{30} \beta$$

where α and β are two prefixed sequences (long 18) with a comparable melting temperature good to perform PCR procedure (for experimental details see Subsection 6.2.1). We assume that each L_i has length 20, as it is a better choice for a primer length, and may not contain CCCGGG as subsequence. The sequences encoding the literals have to be 'very different' to each other, to avoid mismatches during the running of the algorithm.

The main encoding requirements are that the elements of the set $\{X_1, \ldots, X_{30}, Y_1, \ldots, Y_{30}\}$ have no common subwords at least ten long that are complementary to each other, do not begin with sequences of G and do not end with sequences of C long more than five. These conditions are enough to avoid mismatches in the pools involved by the operations of the algorithm, since such a kind of properties are preserved by concatenation and splicing [113].

The length of the initial sequences Z_1 and Z_2 is then 1100bp, resulting from the thirty literals twenty long, the twenty nine occurrences of γ, the prefix and suffix eighteen long. Note that the 2^{30} elements (of the solution space) $\alpha \xi_1 \gamma \xi_2 \cdots \gamma \xi_{30} \beta$ with $\xi_i \in \{X_i, Y_i\}$ have all length 1100.

Algorithm

input $P = \{Z_1, Z_2\}$

1. (Generation step)

for $p = 1, \ldots, 5$ **do**

begin
$P := XPCR_\gamma(P)$
$P := El_{1100}(P)$
end

[2] As an example, we may take γ = CCCCCCCCGGGGGGGGG, having length 16.

2. (Extraction step)

   for $q = 1, \ldots, 126$ **do**

   Consider $C_q = l_i^{(q)} \vee l_j^{(q)} \vee l_k^{(q)}$ and set $P_q := \{L_i^{(q)}, L_j^{(q)}, L_k^{(q)}\}$

   begin

   $P := P \cup P_q$

   $P := H(P)$

   $P := C(P)$

   $P := Taq(P)$

   $P := SmaI_{cut}(P)$

   $P := El_{1100}(P)$

   $P := PCR_{(\alpha,\beta)}(P)$

   end

output P

The output pool P contains the solutions, if there exist, otherwise it is empty, meaning that the instance does not belong to the class of SAT problems.

Although the algorithm is described for an instance of fixed size (in order to suggest a prompt implementation), it can clearly solve instances with any number n of variables and m of clauses (at least theoretically), as long as the generation (first) step is performed $\lceil \log_2 n \rceil$ times and the extraction (second) step m times. In fact, the general form of the algorithm is obtained simply by substituting the value 5, counting the steps of the first cycle *for*, with $\lceil \log_2 n \rceil$, the value 1100, used to perform electrophoresis, with the length of the initial sequences (in this case, as a first instruction a variable should be set with such a length), and the value 126, counting the steps of the second cycle *for*, with the number m of clauses.

Note that the algorithm consists essentially of enzymatic operations, that result convenient with respect to the other methods in terms of efficiency and precision.

5.3.1 Generation step

Firstly we analyze what happens by performing the cycle *for* of the generation step. All the recombinations along the common substrings γ of Z_1 and Z_2 are performed by means of the XPCR method applied to the initial pool. In each of the initial strings there are twenty nine occurrences of γ. The primer γ in the cutting step of the XPCR_γ anneals with one of these occurrences, chosen randomly. If we denote with $\gamma_1, \gamma_2, \ldots, \gamma_{29}$ the occurrences of the same sequence γ, then, after the XPCR cutting step, we obtain from P the two pools P_1 and P_2 respectively:

$$\{\alpha X_1\gamma_1,\ \alpha Y_1\gamma_1,\ \alpha X_1\gamma_1 X_2\gamma_2,\ \alpha Y_1\gamma_1 Y_2\gamma_2,\ \ldots,\ \alpha X_1 \cdots X_{29}\gamma_{29},\ \alpha Y_1 \cdots Y_{29}\gamma_{29}\}$$

and

$$\{\gamma_1 X_2 \cdots X_{30}\beta,\ \gamma_1 Y_2 \cdots Y_{30}\beta,\ \gamma_2 X_3 \cdots X_{30}\beta,\ \gamma_2 Y_3 \cdots Y_{30}\beta,\ \ldots,\ \gamma_{29} X_{30}\beta,\ \gamma_{29} Y_{30}\beta\}$$

The recombination step of the XPCR$_\gamma$ procedure (Section 3.3) applied to $P_1 \cup P_2$ produces a pool with sequences of different lengths (the shorter ones being of kind $\alpha X_1 \gamma Y_{30} \beta$ and the longest ones of kind $\alpha Y_1 \gamma \cdots Y_{29} \gamma X_2 \gamma X_3 \cdots X_{30} \beta$) where only the sequences long 1100 (that is the length of the initial strings) contain just one of each literal (arranged in the sound order).

Therefore, after the selection operation performed by $El_{1100}(P)$, we have a pool with sequences that are product of one recombination of the initial sequences, by means of the null context splicing rule r_γ. Thus the pool includes all the sequences of the following type

$$\alpha X \gamma \cdots \gamma X \gamma Y \gamma \cdots \gamma Y \beta, \qquad \alpha Y \gamma \cdots \gamma Y \gamma X \gamma \cdots \gamma X \beta.$$

Note that after the first step of the cycle *for*, all the possible substrings $L_i \gamma L_{i+1}$ with $i = 1, \ldots, 29$ are present in the pool.

After the XPCR cutting step of the second iteration, we have analogously two pools P_1 and P_2 containing respectively, for $i = 1, \ldots, 28$ and $j = 2, \ldots, 29$:

$$\{\alpha X_1 \gamma_1,\ \ \alpha X_1 \cdots X_i \gamma_i Y_{i+1} \cdots Y_j \gamma_j,\ \ \alpha Y_1 \gamma_1,\ \ \alpha Y_1 \cdots Y_i \gamma_i X_{i+1} \cdots X_j \gamma_j\}_{j>i}$$

and

$$\{\gamma_i X_{i+1} \cdots X_j \gamma_j Y_{j+1} \cdots Y_{30} \beta,\ \gamma_{29} X_{30} \beta,\ \gamma_i Y_{i+1} \cdots Y_j \gamma_j X_{j+1} \cdots X_{30} \beta,\ \gamma_{29} Y_{30} \beta\}_{j>i}$$

After the corresponding recombination step and the electrophoresis selecting the molecules 1100 long, all the possible strings of the following type are present in the pool:

$$\alpha X \cdots X \gamma Y \cdots Y \gamma X \cdots X \gamma Y \cdots Y \beta,\ \alpha Y \cdots Y \gamma X \cdots X \gamma Y \cdots Y \gamma X \cdots X \beta$$

In other words, all the recombinations are generated where, for at most four different values of i, the 'crossing points' $X_i \gamma_i Y_{i+1}$ or $Y_i \gamma_i X_{i+1}$ are present as substrings.

Note that, in the pool resulting after the second step of the cycle *for*, all the possible recombinations $L_i \gamma L_{i+1} \gamma L_{i+2} \gamma L_{i+3}$ with $i = 1, \ldots, 27$ are present as substrings. For example, one obtains the recombination $X_4 Y_5 X_6 Y_7$ as product of the first two steps in the following way:

- the first step produces $\alpha X_1 \cdots X_4 \gamma_4 Y_5 \cdots Y_{30} \beta$ and $\alpha X_1 \cdots X_6 \gamma_6 Y_7 \cdots Y_{30} \beta$ by means of XPCR$_\gamma$ application on the initial sequences along the fourth and the sixth occurrences of γ respectively.
- the XPCR$_\gamma$ of second step recombines the sequences of the pool previously obtained also with respect to the fifth occurrence of γ:

$$\alpha X_1 \cdots X_4 \gamma_4 Y_5 \gamma_5 Y_6 \cdots Y_{30} \beta, \quad \alpha X_1 \cdots \gamma_5 X_6 \gamma_6 Y_7 \cdots Y_{30} \beta$$

$$\downarrow$$

$$\alpha X_1 \cdots X_4 \gamma_4 Y_5 \gamma_5 X_6 \gamma_6 Y_7 \cdots Y_{30} \beta,\ \ldots$$

and $X_4 \gamma Y_5 \gamma X_6 \gamma Y_7$ is obtained as substring of product strings.

At this point it is clear how after the k-th step of cycle *for*, all the possible recombinations of 2^k consecutive literals are present in the pool as substrings. Since after the generation step all the possible recombinations of n literals have to be present, then $\lceil \log_2 n \rceil$ steps suffices to produce the solution space.

5.3.2 Extraction step

Once obtained the solution space, in the extraction step the (sequences encoding the) *true* solutions are selected from (those encoding) the *possible* solutions. Its implementation combines a polymerase extension with a cut of the restriction endonuclease SmaI, and after an electrophoresis the sequences of interest are amplified by a standard PCR.

The key process lies in the steps corresponding to H, C, and Taq operations, which recall those of a standard PCR but produce a different result depending on the type of 'primers'. We have three primers instead of two, and overall they are designed to anneal with a same side of the double strands.

To each step of the cycle *for* is associated a different clause, and after that step all the sequences satisfying that clause have been extracted. The iteration of such a step for all clauses allows the final pool to contain only the strings (encoding assignments) that satisfy *all* the clauses.

Let us explain the computation of the extraction phase with an example. We consider the process corresponding to the first step of the for cycle, namely related to the clause $C_1 = x_{10} \vee \neg x_{27} \vee x_{15}$. It starts on the pool P containing the solution space, and firstly it sets $P_1 = \{X_{10},\ Y_{27},\ X_{15}\}$ as indicated by C_1.

1. $\mathbf{P} := \mathbf{P} \cup \mathbf{P_1}$
   Many copies of *literal primers*, that are the elements of P_1, are added to the pool (also free nucleotides and PCR ingredients actually, but this is not relevant from the algorithmic point of view).
2. $\mathbf{P} := \mathbf{H}(\mathbf{P})$
   When the pool is heated then a denaturation occurs (see Figure 5.3 from [71]). For each molecule in the pool the two strands come apart without any break, because the hydrogen bonds between complementary nucleotides are much weaker than the covalent bond between adjacent nucleotides along a strand.

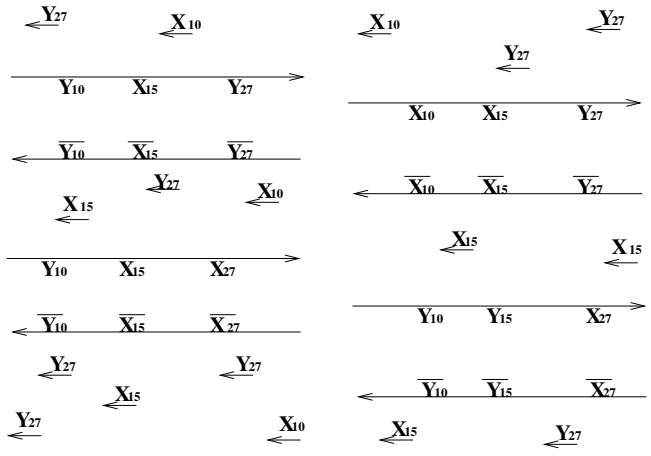


Fig. 5.3. Denaturation of all the molecules.

3. $\mathbf{P} := \mathbf{C}(\mathbf{P})$
   By cooling down the pool, literal primers bind to their complementary strands, as in Figure 5.4 (which is taken from [71]).

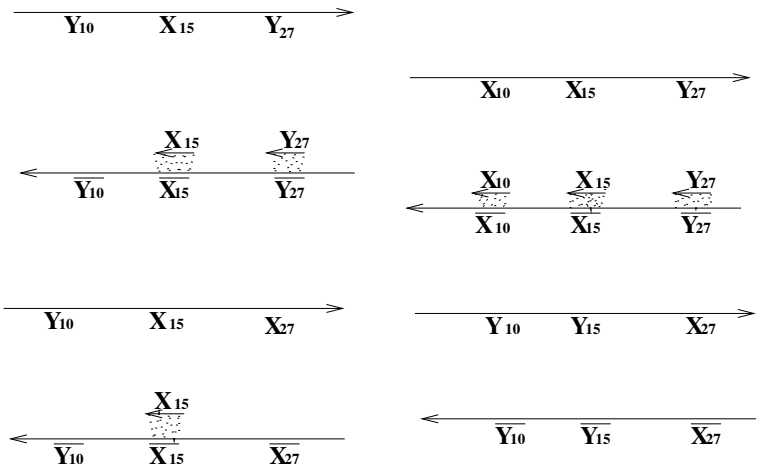


Fig. 5.4. Selecting by literal primers.

4. $\mathbf{P} := \mathbf{Taq}(\mathbf{P})$
   Polymerase extension elongates the double literal primers as in Figure 5.5 (taken from [71]), where the resulting single strings are shown in a dots frame.

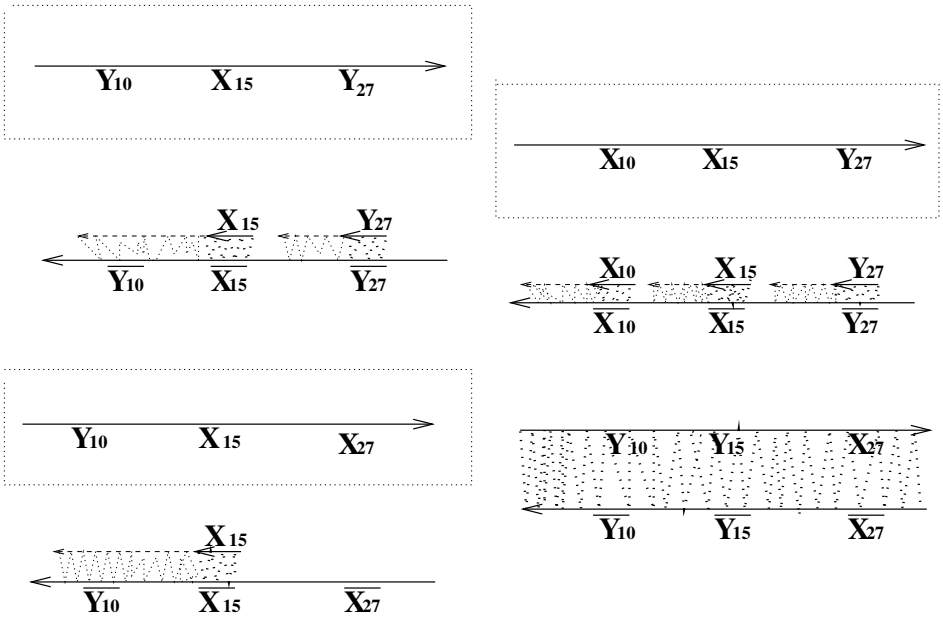


Fig. 5.5. Detecting of the 'good' assignments as single strings.

5. $\mathbf{P} := \mathbf{SmaI_{cut}}(\mathbf{P})$
   Since the restriction site of SmaI is included (only) in γ, which is uniformly distributed along the molecules, and since the enzyme cuts only double stranded molecules, then after this operation only the single strings have length 1100. They encode assignments satisfying the clause C_1 because contain at least one of the literal primers in P_1.
   The algorithm selects these single strings by means of the operation $\mathbf{El_{1100}}(\mathbf{P})$, and then amplifies them with a $\mathbf{PCR}_{(\alpha,\beta)}(\mathbf{P})$ which restores their double stranded form[3].

The strings extracted by the first step are then filtered by the second step in such a way that only those satisfying also the clause C_2 are retained, and so on for all the 126 clauses.

Every solution is present in the final pool, because each primer (given in many copies) surely binds to some of its complementary regions, thus the algorithm is complete. Instead encoding mismatches could cause non-correctness; in fact, if a literal primer anneals within a wrong location, then that string is selected as good assignment even if it does not. But each solution is filtered m times, and to have a non-solution assignment in the final pool, mismatch errors should occur (selecting it) for all the clauses which it does not satisfy, and this case is quite rare.

This algorithm has been not yet experimented, thus many pitfalls could arise in the laboratory, that do not appear in the above design. However, a successful implementation of the first part of the algorithm just would improve the results obtained for the XPCR generation algorithm (presented in Chapter 4). While the laboratory implementation of the second part could show that the extraction step can be performed by using only the principles of polymerase extension and blunt cuts by a restriction enzyme. With respect to [28] for example, the procedure would result simpler, as no chemical synthesis and any *ad hoc* automated thermocycler are required.

5.4 A mutagenesis algorithm

Mutagenesis is a very important methodology which substitute any occurrence of a substring γ with another string δ, within the strings of a given pool. In the special case where $\alpha = \lambda$ we have 'insertional mutagenesis', that is used to block the expression of a gene because produces 'mutations with lost of functionality'. In general, mutagenesis in vitro of cloned genes is the standard tool in the functional analysis of polymers as DNA and proteins. It may be 'casual' or 'situ-specific' (that is site directed) mutagenesis, the first one generates mutations wherever, it is used as a first approach to identify a functional region in a given fragment, and the second one is used in a second phase to analyze the functionality of a specific region by altering its sequence. Traditionally the techniques used to perform mutagenesis are restriction enzymes or other kinds of enzymes, oligos synthesize, hybridization and sequencing.

[3] From an experimental point of view, it could be better to perform electrophoresis on double strings. For this case, electrophoresis step can be exchanged with the PCR amplification.

In the following we present a mutagenesis algorithm based on XPCR procedure that was introduced by Manca et al. in [150], and was implemented by the experiment reported in Section 6.5. Other more or less similar algorithms for mutagenesis may be found in DNA computing context, in [105] and [131] respectively.

Given in input the pool $P = \{\alpha\gamma\beta\}$, the algorithm replaces any occurrence of a string γ with an occurrence of the string δ, by producing in output the pool $P = \{\alpha\delta\beta\}$. We recall that the factor of a string α occupying from its n-th position to its m-th position, with $m \geq n$ (and both smaller than or equal to the length l of α), is usually denoted by $\alpha[n, m]$. One may refer to the same faction by with negative indexes as $\alpha[n - l, m - l]$, meaning that we count the positions in the backward order, from the right side to the left one of the string. Let $P = \{\alpha\,\gamma\,\beta\}$ and $Q = \{\alpha[-18, -1]\,\delta\,\beta[1, 20]\}$.

Mutagenesis(P, γ, δ) Algorithm

input P, Q

1. **split** P into new P_1 and P_2 (with the same approximate size);
2. **perform** $P_1 := PCR_{(\alpha[1,18],\alpha[-18,-1])}(P_1)$ **and** $P_2 := PCR_{(\beta[1,20],\beta[-20,-1])}(P_2)$;
3. **select** $P_1 := El_{|\alpha|}(P_1)$ **and** $P_2 := El_{|\beta|}(P_2)$;
4. **mix** the first of the obtained pools with Q, that is $P_1 := P_1 \cup Q$;
5. **perform** $P_1 := PCR_{(\alpha[1,18],\beta[1,20])}(P_1)$;
6. **select** $P_1 := El_{|\alpha|+|\delta|+20}(P_1)$;
7. **mix** $P := P_1 \cup P_2$;
8. **perform** $P := PCR_{(\alpha[1,18],\beta[-20,-1])}(P)$;
9. **select** $P := El_{|\alpha|+|\beta|+|\delta|}(P)$;

output P.

If we call α_1, α_2 respectively the prefix and the suffix 18bp long of α, and β_1, β_2 respectively the prefix and the suffix of β with length 20bp, the XPCR Mutagenesis Algorithm may be depicted as in Table 5.1.

The ability to easily alter a DNA sequence allows to modify the structure of the gene products: RNA and proteins. Mutagenesis technology has changed the way to study molecular biology, by creating the concept of 'inverse genetics': first one modifies the genetic sequence and then one analyzes the functionality. A consequence of this approach is the possibility to produce modifications in the natural genetic products in such a way that they work with a better performance. In fact, protein engineering receives wide interest from the industry, especially for a likely manipulation of antibodies, that are known in detail and have an enormous potentiality for clinical applications.

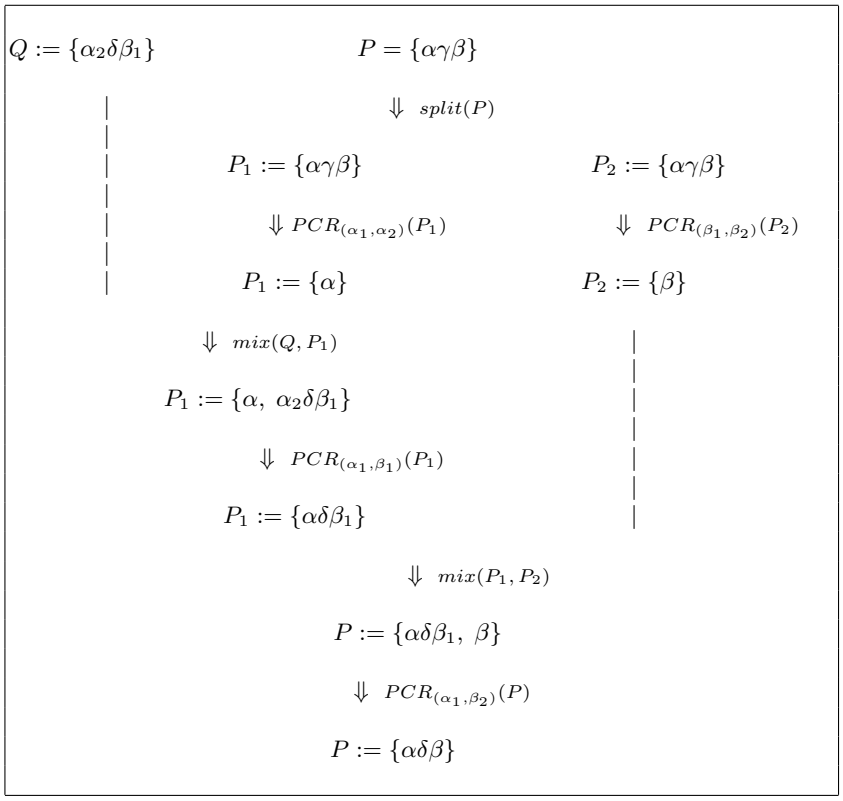


Table 5.1. XPCR Mutagenesis Algorithm transforming a pool $\{\alpha\gamma\beta\}$ into a pool $\{\alpha\delta\beta\}$, where $\alpha = \alpha_1 \cdots \alpha_2$ and $\beta = \beta_1 \cdots \beta_2$.

5.5 Past and future applications

In this section I would like to emphasize the impact that PCR and other biotechnologies (such as cloning) have had in society and in other contexts than DNA computing, and to suggest on the way (hypothetical) future applications that PCR based methods could have. A news article by Jean Marx quite soon documented the story of pivotal development of PCR [163]. In the following we report some of its main topics, together with notions and argumentations that may be found in [231].

Many areas of biomedical and clinical microbiology research depend on the analysis of uncommon variations in individual genes or transcripts. Modifications of genetic sequences, such as substitution of a substring with another one, insertion or deletion of substrings, are called *mutations*. The identification and the quantification of rare mutations are objects of wide interest, as well as the study of variations in gene sequences or transcripts in specific populations or in cellular tissues [53]. Mutations are responsible of hereditary diseases and other diseases like cancer, which is just implied by genetic alterations. Namely, the existence of genes capable to stop the tumored cells were observed together with chromoso-

mal deletions, and such observations suggested that cells contain genes suppressor of growth that have to be inactivated before the cancer can develop. Moreover, deletions of esons are exploited for the direct diagnosis of muscular dystrophy of Duchenne. Since the discovery of PCR, to detect or extract given DNA sequences (such as mutations) is possible and easy.

Mutations are important also because they represent the origin of the phenotype varieties, on which is based the natural selection leading the evolution. Tissue from Egyptian mummies, or from a woolly mammoth, who last walked the Siberian steppes some 40.000 years ago, would not seem to be a likely source of DNA for sequence analysis. After all, DNA breaks down very readily, and even though the mammoth had been frozen in the Siberian ice until it was chipped out, its DNA had undergone extensive degradation. With the aid of PCR though, researchers were able to produce enough mitochondrial DNA to determine the nucleotide sequence, because the DNA to be amplified by PCR does not have to be in good shape. The results showed, not surprisingly, that the mitochondrial DNA of this extinct species is very similar to that of the modern elephant, its close relative. Variations in DNA structure were also used to trace evolution of human populations. For example, a particular mutation was identified, a 9bp deletion, in human mitochondrial DNA that apparently originated in East Asia. It can aid in tracking the migrations of ancient peoples, namely, the finding of the 9bp deletion in some American Indians and in New Guineans indicates that migrants form East Asia contributed to the founding of these populations.

This opened the way to a variety of applications of PCR technique in addition to analyzing ancient DNA samples. One which has advanced very rapidly was the forensic use of the technology to identify (or exonerate) suspects in rape and murder cases. The biological evidence found at the scene of violent crimes, including bloodstains, semen, and hair, is often of such a poor quantity or quality that it cannot be analyzed by other forensic techniques. The DNA is thus amplified and then compared with the suspected person's DNA to obtain an identification. Moreover, polymorphic loci exist along the protein polymers that can be uniquely associated to a single individual (the test is called *DNA fingerprinting*).

PCR was also applied in clinical medicine as a gene amplification technique. It is used, for example, to simplify and speed up the prenatal diagnosis of genetic diseases, including β-thalassemia and sickle cell anemia, to detect viral infections, including AIDS, to do the tissue typing needed for matching the donors and recipients of transplanted organs, and to identify the genes that confer susceptibility to autoimmune diseases. These diseases, which are caused by an abnormal attack of the immune system on the body's own tissues, actually are not classified as genetic diseases. They have a genetic component, however, in that susceptibility to many of the diseases has been linked to particular variants of a specific class of molecules, that mediate the immune cell interactions necessary for mounting normal immune responses. By allowing the rapid amplification and sequencing of genes, the PCR technique also facilitated the studies needed for a better understanding of genetic contributions to autoimmunity.

Still in immunological context, DNA generation methods could be important to study rearrangements of antibody genes. Some organisms have enzymes that are able to recombine the 'ipervariable' regions of antibodies in such a way that

cells can defend from an external attack of pharmacies or of virus. This process is crucial to understand how the immunological response works and how cells develop a specific resistance to some pharmacies. Experiments have been carried out, where substrates generated by artificial recombination were put in cells, such as fibroblasts and lymphocytes B, then treated with some pharmacy. The surviving cells have been those capable to develop a resistance to the pharmacy. Such a resistance was induced by the enzyme able to recombine the artificial substrate. Finally, from human lymphocyte B was extracted, cloned, and studied, the DNA segment responsible of such a recombination, and it resulted to contain two special genes that cooperate to activate the recombination of antibodies' portions with high efficiency.

XPCR extraction on the other hand, has the potentiality to be really useful to detect mutant gene segments. For example, in some forms of hereditary cancer, cloned genes may be used to determine whether the components of a family are under risk to have cancer. An extraction method applied on the DNA from cells of the familiars could point out the presence of a specific gene. The high sensitivity of the PCR, and the extraction methods based on PCR, also make it well suited for detecting viruses, especially those viruses that go underground in infected persons, often persisting for several years in a small number of cells before causing obvious symptoms of disease. Human immunodeficiency virus type I (HIV-1), the AIDS virus, is a case in point. AIDS has a latent period of several years, and very few susceptible cells (perhaps 1 in 10.000) actually carry the virus. Nevertheless PCR is used to amplify HIV-1 gene sequences in the DNA of peripheral blood cells from infected individuals, thereby allowing the direct detection of the viral presence; it was shown that PCR is able to pick up one to ten copies of viral DNA per 1 million of cells!

PCR technique, by making easier the detection of HIV-1, facilitates the epidemiological studies needed to resolve the issue of whether the virus persists in all antibody-positive individuals, or whether it may sometimes be eliminated by the immune system. For clinical purposes, it is possible to use PCR to estimate how much HIV-1 a person carries. Individuals with the larger virus burdens, who are presumably at high risk of coming down with AIDS, might then be candidates for early drug therapy. In addition, the technique might help to determine whether a candidate antiviral drug actually reduces the mount of virus carried by patients undergoing therapy.

5.5.1 Support to cancer research

The study of recombinant DNA has been essential in the progress for the diagnosis of cancer. PCR is also aiding efforts to track down the cellular changes that play a role in cancer development. For example, the normal genes that control cell growth and differentiation may sometimes undergo alterations that convert them to cancer-causing genes, called oncogenes. For some genes, such as *ras*, a change in a single base pair is sufficient to do this. Determining how frequently *ras* gene activation might contribute to the development of human cancers has been difficult. Although *ras* oncogenes have been found in several types of human cancers, in most studies, the percentage of tumors carrying the active oncogene was low, 10% or

less. This figure may have been so low at least partly because identifying point mutations, such as those causing *ras* gene activation, could be difficult. While using PCR, more than a third of human colon cancers was shown carry oncogenic *ras* mutations. Similar approaches showed that cancer cells from about one quarter of patients with acute myelitis leukemia also contain *ras* oncogenes.

When patients undergo surgery for cancer, samples of the tissue removed are embedded in paraffin blocks. This is done so that thin tissue slices can be made and then examined by a pathologist to confirm the cancer diagnosis. The blocks of paraffin-embedded tissue are usually kept so that there may be a worldwide collection of such preserved tissue specimens, with defined pathologies and known clinical results for the patients. DNA sequences of the human papilloma virus was detected in samples of cervical cancer tissue with the aid of the PCR, even though the tissue preparations had been made years ago and the DNA is likely to be degraded. The most important consequence of this work was that it opened up the possibility of using the tissue specimen collection for studies aimed at tracing the contributions of viruses and oncogenes to cancer development.

5.5.2 Ethical aspects

In 1971, while the first genetic cloning experiments were being made, some anxiety arose about the safety of such a kind of experiments. Communities were worried especially about the cloning of genomes of tumored virus and the replicating of the molecules in bacterial cells such as *Escherichia Coli*, because of the possibility to transmit human cancer by means of these bacteria. At that time though, they did not have the data necessary to evaluate the actual risks[4].

The signatories of the 1974 Berg letter to *Science*[5] had made a principled public appeal for biologists to hold off doing two specific types of recombinant experiment and to weigh the possible consequences of others, until the corresponding risks will have been clearer. Such an evaluation was done in February of 1975 by more than 100 famous molecular biologists, in the Conference Center of Asilomar, at Monterey, California. The issues at Asilomar were clearer than those that arose later. They gathered many proposals, and one of them was to use only genetically enable bacteria to clone DNA, so that they could grow only in vitro[6]. These directives were followed and formalized by a special committee nominated by National Institutes of Health (NIH) in USA, that in particular prohibited experiments on genes from tumored virus. In July 1976 the current indications were rectified by a Recombinant DNA Advisory Committee (RAC) constituted also by non-scientists and lowers. The European formulation of such a mobilization was the Genetic Manipulation Advisory Group (GMGA), in England. More data and further ex-

[4] M. Singer and D. Soll. *Guidelines for DNA hybrid molecules*, Science, 181:1174, 1973.

[5] P. Berg, D. Baltimore, H. W. Boyer, S. N. Cohen, R. W. Davis, D. S. Hogness, D. Nathans, R. Roblin, J. D. Watson, S. Weissman, and N. D. Zinder. *Potential biohazards of recombinant DNA molecules*. Science. 185(148):303, Jul 26, 1974.

[6] P. Berg, D. Baltimore, S. Brenner, R. O. Roblin, and M. F. Singer. *Asilomar conference on recombinant DNA molecules*, Science, 188:991-994, 1975.

periments convinced the NIH to admit the cloning of tumored virus genes, in January of 1979[7].

Nowadays research performed by DNA manipulation techniques is extraordinarily growing. The application to the pharmacy production and vaccines gave birth to the biotechnological industry. The technological progress about genetic cloning, genetic structure and expression, and human genetics, of the last twenty years has no comparison with the knowledge we had accumulated in the previous times, and PCR was definitely a milestone in such an advancement.

Synthetic proteins are made for facing different pathologies, like cancer, allergies, autoimmune diseases, neurological disturbs, cardiopathologies, hematological alterations, genetic infections and diseases, production of soap and food! Very interesting examples of recombinant pharmacy produced by genetic engineering are the human insulin, which is the hormone regulating the sugars metabolism, the human Growth Hormone, the Hepatitis B vaccine, and a rational projecting of pharmacies.

Section 5.1 and Section 5.3 include respectively more or less wide parts of the following papers:

- G. F., *Hard Instances of SAT*, TR of the project 'Biomolecular algorithms for NP-Complete problems', Department of Computer Science, University of Pisa, July 2002.

- G. F., *A Polymerase Based Algorithm for SAT*, M. Coppo and E. Lodi and G. M. Pinna, editors, Theoretical Computer Science, Lecture Notes in Computer Science 3701, Springer, pp 237-250, October 2005.

[7] Department of Health, Education, and Welfare. *Legislation, Regulations, or Statutes for Previous U.S. Bioethics Initiatives*, Federal Register, January 11, 1979.

6

Laboratory Experiments

There is no higher or lower knowledge, but one only,
flowing out of experimentation.

Leonardo da Vinci

Three laboratory experiments validating respectively the extraction (Section 3.4), recombination (Section 4.1) and mutagenesis (Section 5.4) algorithms are reported in the last sections. The first two ones have been designed and directed entirely by myself, the third one just partially (at that time I was spending few months at University of South Florida). I have not used any design service for the encodings, just ad hoc variants of results from the literature (for example, [8, 29, 114]) and some checking by means of a program, which given a couple of sequences finds common subsequences having length greater than five. This has been an useful information to avoid mismatches, especially in our cases where PCRs have been often performed at low (under suggested limits) temperatures.

The trials are presented according to their chronological order, so that the reader can notice as the experimental tricks (about preferred lengths, melting temperatures and annealing specificity) and the tested sequences from one experiment were rearranged for the next one. The experiments were manually carried on by Cinzia Giagulli, who is a postdoc researcher from the Department of General Pathology of University of Verona. Initially I was thinking that the main difficulty to set up an experiment with her would have been to find a common language to exchange ideas, a simple manner to explain an algorithm and its (theoretical) implementation and to understand the technical ways to perform the required operations. But quite soon I have crashed into a totally different reality, and I ended up with as unexpected as practical problems. Section 6.2 reports what I noticed during my visits in the laboratory. Apart mismatching problems studied in the literature and partially described in Section 3.5, indeed many other (more practical) problems can be avoided by 'good' encodings design. One studies on the books about the difference between the (weaker) bonds A-T and the (stronger) bonds C-G, but usually he/she does not realize how much this difference influences the melting temperature of the sequences (connected to the percentage of C-G base

pairs) and the formation of secondary structures. One studies as well that the agarose gel for electrophoresis can be made more dense in order to discriminate the shortest strings, but there are not reported the manual difficulties to manage a 5% gel with respect to a 3% one. These are just examples of the fact that there have been a lot of factors hidden behind the passage from the algorithms explained in the previous chapters to their laboratory implementation described in the following.

Firstly, I realized that after the theoretical design of a DNA algorithm and before starting to talk about implementation, one has to submit the algorithm into the following constraints:

- Maximum length of (initial) sequences to order from a company has to be 150bp. In our case sequences shorter than or equal to 100bp have been finally preferred, while very long sequences to work with have been extracted from human gene RhoA.
- Lengths of primers (to use during PCRs) have to be from 17 to 30 bp. Currently we are doing an experiment with primers 20 long, that's the best choice.
- Primers should have a balanced distribution of C-G and A-T base pairs. They need to be with no internal secondary structure and with no complementarity between their $3'$ ends (otherwise they anneal each other, forming 'primer-dimers').
- Primers must have comparable melting temperatures (ranging from 55 to 70 Celsius degrees), and they should start and end with a C or a G, in order to allow specific annealing.
- All sequences that are separated by electrophoresis should have length at least 50bp

The first constraint is related to economic and attendance time reasons, the last one to the feasibility to work with gel 3%, and the middle ones to biochemical factors. In most PCR applications, it is the sequence and the concentration of the primers that determine the overall assay success. Melting temperature of primers is thus something to take strictly in account during the encoding phase in order to perform specific PCRs during the experiment. Section 6.1 explains with more details how it is calculated.

6.1 Melting temperature

By definition, the temperature at which 50% of a given oligonucleotide is hybridized to its complementary strand is called *melting temperature*, shortly Tm.

The melting temperature characterizes the stability of the double DNA filaments formed between an oligonucleotide and its complementary strand. It is very important for determining the optimal temperature at which to use an oligonucleotide as a primer in PCR applications (annealing temperature), or in general as a probe for in situ hybridization.

In the absence of destabilizing agents, like formamide or urea, Tm will depend on three major parameters:

1. *The sequence*: a GC-rich sequence has a higher melting temperature.
2. *The strand concentration*: high oligonucleotide concentrations favor paired formation, which results in a higher melting temperature.
3. *The salt concentration*: high ionic strength results in a higher Tm because cations stabilize the DNA duplexes.

Many methods have been proposed to empirically determine Tm values. For oligonucleotides s having length at most 20 bases, we used the following classical calculation for melting temperature, according to the Wallace rule for short oligonucleotides

$$Tm(s) = 2(|s|_A + |s|_T) + 4(|s|_G + |s|_C)$$

where $|s|_N$ denotes the number of occurrences of the symbol N in the sequence s.

In the case of longer DNA fragments, the nearest-neighbor method [30, 212], which combines solid thermodynamics and experimental data, offers a reliable estimation of the Tm. In the nearest-neighbor formula, the two parameters affecting the Tm value are the salt concentration and the concentration of single stranded DNA. These two experimental variables are here chosen in order to optimize DNA amplification by PCR and their values are fixed at 50 mM (millimoles) of salt and 250 pM (picomoles) of oligonucleotide, as suggested in [205].

The formula to calculate the Tm by the nearest-neighbor method is

$$Tm = H(A + S) + R\ln(Ct/4)273.15 + 16.6\log(P),$$

where

- H is the sum of the nearest-neighbor enthalpy changes for hybrid formation (it is a negative value, estimated in cal per mole).
- A is a constant for helix initiation. It is equal to -10.8 (cal. per mole) for non-selfcomplementary sequences and - 12.4 for selfcomplementary sequences.
- S is the sum of the nearest-neighbor entropy changes for paired formation (it is a negative value, estimated in cal per mole).
- R is the molar gas constant (1.987 cal per mole).
- Ct is the total molar concentration of strands when oligonucleotides are not self complementary. It is equal to 4 times this concentration in the case of selfcomplementary sequences.
- P is the salt concentration.

The Tm value is calculated for phosphodiester oligonucleotides used in DNA-amplification experiments. Even though this calculation does not take into account any nucleotide modification, like biotin, digoxigenin, fluorescent dyes, it still gives us a good estimate of the starting conditions.

Coming back to our experiments, the melting temperatures of the primers involved in each PCR have been made close each other, and the corresponding PCR was then performed by setting the lowest temperature in order to make more specific the primer annealing step.

6.2 In a laboratory

Let us assume for example that we want to implement the following simple algorithm, starting from a pool of 10^{11} molecules, with four kinds of sequences S_1, S_2, S_3, S_4 uniformly distributed, having the same length (100bp), the same prefix α and the same suffix β.

1. Take randomly 100 molecules
2. Amplify them by $PCR(\alpha, \beta)$
3. Check the amplification product by electrophoresis.

First of all we have to set up a solution containing 10^{11} molecules. We have the test tubes arrived from the company, each of them contains a different sequence from $\{S_1, \overline{S_1}, S_2, \overline{S_2}, S_3, \overline{S_3}, S_4, \overline{S_4}\}$, with a concentration of 100 μmoles (a mole is 10^6 micromoles) per liter. Since we can take just prefixed quantities of solution by means of tared pipettes[1], some counts are necessary in order to know how to take 10^{11} and then 100 molecules from a solution.

Since the sequences have to be uniformly distributed we need to take $10^{11}/8=$ 12,5 10^9 molecules from each test tube. The following proportions ($6,022\,10^{23}$ is the Avogadro number) indicate (by X) how many micromoles we have to take of each sequence in how many microliters (Y) of solution.

$$1\text{mole} : 6,022\,10^{23} \text{ molecules} = X : 12,5\,10^9 \text{molecules}$$
$$X = 2,075\,10^{-14} \text{ moles} = 2,075\,10^{-8} \mu\text{moles}$$

$$100\,\mu\text{moles} : 1L = 2,075\,10^{-8}\mu\text{moles} : Y$$
$$Y = 2,075\,10^{-10}L = 0,0002075\,\mu l$$

Y is definitely lower than 0.2 μl, which is the minimum we are able to take with a pipetman. If we put (by a P2 pipette for example) 1 μl of each sequence into 100 μl of sterile water, and from this solution we transfer 1 μl into 100 μl of sterile water, then we can take (by a P10) 2,075μl from the solution that corresponds to 12.5 10^9 molecules of each sequence. Here we have preferred to dilute two times

[1] Main tools to handle liquids.

1. Accurate and precise, ideal for molecular biology techniques (PCR, DNA sequencing)
   - P2 (capturing from 0.2 μl to 2 μl)
   - P10 (capturing from 1 μl to 10 μl).

2. Suited for small-volume dispensing of aqueous fluids of moderate viscosity and density (molecular biology, immunology)
   - P20 (capturing from 2 μl to 20 μl)
   - P100 (capturing from 20 μl to 100 μl)
   - P200 (capturing from 50 μl to 200 μl)
   - P1000 (capturing from 200 μl to 1000 μl).

3. Ideal for large-volume dispensing of aqueous fluids of moderate viscosity and density.
   - P5000 (capturing from 1000 μl to 5000 μl)
   - P10ml (capturing from 1 ml to 10 ml).

in 100 μl instead of diluting once in 1000 μl and then taking just the minimum possible (around 0.2 μl). This is the experimental way to proceed, one has always to be aware that we are dealing with inexact estimations and instruments.

By taking 2,075μl from the solution corresponding to every sequence, we reach 16.6 μl of solution which we round by adding 993,4 μl of sterile water. Thus we obtain 10^{11} molecules uniformly distributed in $1ml$ of solution. To select randomly 100 molecules starting from 10^{11} molecules in 1000 μl, it is enough to dilute analogously as above, by taking 1 μl from the current solution and putting it into 1000 μl of water for two times. Next step is to amplify such 100 molecules (with a same prefix α and a same suffix β) by polymerase chain reaction.

6.2.1 PCR protocol

PCR is an in vitro method for enzymatically synthesizing defined sequences of DNA. The reaction uses two oligonucleotide (called primers), that hybridize to the opposite strands of the DNA which has to be amplified in their terminal parts. The elongation of the primers is catalyzed by Taq enzyme, which is a heat-stable DNA polymerase that is isolated from the thermophiliac eubacterium Thermus aquaticus.

PCR was originally performed with a DNA polymerase from the bacterium *Escherichia Coli*, but this enzyme is inactivated by the high temperatures needed for strand separation, whereas the primary requirements for a polymerase used in the PCR are good activity at temperatures around 75°C and ability to retain that activity after prolonged incubations at even higher temperatures (95°C). There are at least three DNA polymerases (Taq, Tth, Pwo) which meet those requirements, but Taq is the most efficient. It has a half life of less than 5 minutes at 100°C, however, the enzyme retains its activity for much longer times (half life up to 40 minutes) at 95°C. The high processing, absence of exonuclease activity, and temperature optima of Taq DNA Polymerase make the enzyme useful even for DNA sequencing, especially where the resolution of secondary structures plays a major role. Moreover, this Polymerase has the advantage to accept modified deoxyribonucleotide tri-phosphates as substrates, and can be used to label DNA fragments either with radio nucleotides, digoxigenin, biotin, or fluoresce.

A repetitive series of cycles involving template denaturation, primer annealing, and extension of the annealed primers by Taq DNA Polymerase results in exponential accumulation of a specific DNA fragment. In fact, since the primer extension products synthesized in a given cycle can serve as a template in the next cycle, then the number of DNA copies approximately doubles every cycle: thus, 20 cycles of PCR yield about a million copies (2^{20}) of defined DNA.

There exist standard protocols to perform PCR, however, in principle, each physical and chemical component can be modified to produce a potential increase in yield, specificity or sensitivity, and these factors are not independent of each other. It follows a description of how a PCR(α, β) was performed in our laboratory.

We have a termocycler to set, an electric vortex to mix solutions, a refrigerator containing the test tubes of the DNA sequences and of the PCR ingredients, and all the pipettes and differently sized test tubes within easy reach. We start by cleaning

the counter carefully, and by wearing fresh gloves. In fact, the ability of the PCR to amplify a single molecule means that trace amounts of DNA contaminants could serve as templates, resulting in amplification of the wrong template (false positive).

Firstly, the PCR mixture has to be prepared for three[2] samples of 50 μl, with the following ingredients.

- *Buffer of PCR* (10x, meaning that it is initially 10 times concentrated). It is a transparent liquid having saline concentration and PH such that the physiological environment of the cell is simulated, thus the enzyme can work as usual.
- *Taq DNA Polymerase*. For most assays, the optimum amount of it is between 0.5 and 2.5 units (we used 1.25 units). Increased enzyme concentrations sometimes lead to decreased specificity.
- *dNTP mixtures*. If the concentration of the four dNTPs is imbalanced, this can reduce Taq fidelity. We used an amount of 200 μM.
- $MgCl_2$. It is a necessary ingredient to catalyze the enzyme action, and its optimal concentration may vary from approximately 0.5 mM to 5 mM, the standard value is 1.5mM. Magnesium influences enzyme activity, increases the Tm of double stranded DNA, and forms soluble complexes with dNTPs to produce the actual substrate that the polymerase recognizes.
- *Primer concentrations* between 0.1 and 0.5 μM are generally optimal (we have taken 0.5 μl of α and 0.5 μl of β). Higher primer concentrations may promote mis-priming and accumulation of unspecific product. Lower primer concentrations may be exhausted before the reaction is completed, resulting in lower yields of desired product.
- *DNA template*. The primer/template ratio strongly influences the specificity of the PCR and should be optimized empirically. With a small amount of template the primers may not be able to find their complementary sequences, whereas too much template may lead to an increase in mis-priming events. The purity of template also influences the outcome of the reaction.

We used ice to which insert test tubes during the preparation of such a mixture, in order to avoid unspecific formations, and pipettes instead of vortex to mix the solution once the Taq was in the solution (otherwise foam could be produced). The PCR program was then set as in the next points.

- Initial denaturation (at temperature 95° C).
  It is very important to denature the template DNA completely. Initial heating of the PCR mixture for few minutes (we set 4) at 95° C is enough to completely denature complex genomic DNA, so that the primers can anneal to the template as soon as the reaction mix is cooled. If the template DNA is only partially denatured, it will tend to 'snapback' very quickly, preventing efficient primer annealing and extension or leading to 'self-priming', which can lead to false positive results.

[2] We take two extra samples called *control reactions*: the negative one, which contains all reaction components except the primers, and the positive one, which contains all reaction components with a different DNA template. They are useful to check that the reaction components are good and work well, in fact, no amplification is expected for the negative control, and a different amplification is expected for the positive control.

The denaturing step is performed (at temperature 95°C) initially for longer time, and then repeated for some seconds (we set 30) at the beginning of every cycle, to avoid that an incompletely melted DNA snaps back, thus giving no access to the primers.

- Primer annealing.
  Temperature for this step has to be optimized empirically, we have chosen to set the lowest between the melting temperatures of the primers α and β (that is calculated as explained in Section 6.1) for one minute.
  The choice of the primer annealing temperature is probably the most critical factor in designing a high specificity PCR. If the temperature is too high, no annealing occurs, but if it is too low, unspecific annealed products increase dramatically. If a primer has a $3'$ end which anneals to any site on a template, irrespective of whether it anneals specifically or not, the primer will be elongated from that $3'$ end. Also, at low temperatures, primers may hybridize partially without involving their $3'$ end, and such hybrids will not be extended.

- Primer extension.
  It is normally carried out at temperature 72° C, that is ideal for Taq DNA Polymerase working (at 72° C it can add 60 bases per second!). Usually a 20 second extension is sufficient for fragments shorter than 500bp and a 40 second extension is sufficient for fragments up to 1.2Kb. However, these times may vary (for sequences 100 long we set 10 seconds as duration of this step).

- Cycles number.
  In an optimal reaction, less than 10 template molecules can be amplified in less than 40 cycles to a product that is easily detectable on an ethidium bromide stained gel (see Section 6.2.2). Most PCRs should, therefore, include only 25 to 35 cycles. As cycle number increases, unspecific products can accumulate.

- Final extension.
  Usually, after the last cycle, the reaction tubes are held at 72° C for 5-15 minutes to promote completion of partial extension products and annealing of single stranded complementary products.

From a biotechnological point of view, main limitations of PCR are:

- One only obtains a DNA fragment. To see this DNA at work inside a living organism, some type of cloning has to be done.
- Only relatively short sequences can be amplified reliably. Anything more than 10,000 base pairs is unlikely to be amplified.
- One needs to know the right primer sequences to use, at both ends of the sequence you want to amplify. If two related genes have the same end sequences, one might amplify the wrong gene.
- The possibility of mis-incorporation of dNTP, which arises from the polymerase characteristics, must be taken in consideration.

Let us assume we are at the end of the second step of the algorithm: we just amplified by 35 cycles of PCR(α, β) 100 molecules containing four kinds of se-

quences S_1, S_2, S_3, S_4 uniformly distributed (having the same length, the same prefix α and the same suffix β).

Before of carrying on an electrophoresis to verify that the amplification happened correctly (we expect to see amplified sequences all 100 long), we check that our amount of DNA is enough to be detected by electrophoresis. On an agarose gel, the limit of visibility is 50ng of DNA, and we want to be sure to put, say, 100ng of DNA in the gel.

We recall that 100 ng $= 10^{-7}$gr, and that the number of moles of a substance is given by the rate of its weight to its molecular weight. The molecular weight of each DNA sequence is known, it depends on that of its bases and on its length. In fact, the molecular weights of the four nucleotides are not identical (phosphates and sugars are a constant but bases have different weights), but in routine work we usually average the molecular weights of all four bases and go with 324.5 Dalton per DNA nucleotide (that means 324.5 grams/mole of individual nucleotides[3]), and a corresponding base pair weight of 649 Dalton. For our sequences S_1, S_2, S_3, S_4, we computed the molecular weight average of $246700 : 4 = 61675$ g/m.

$$
\begin{aligned}
&10^{-7}\text{g} : 61675\ \text{g/m} = 1,621403\,10^{-12}\ \text{moles}\\
&1\text{mole} : 6,02210^{23}\ \text{molecules} = 1,62140310^{-12}\ \text{moles} : X\\
&X = 9,76408886610^{11}\ \text{molecules}
\end{aligned}
$$

These counts allow to know that 100 ng of DNA approximately correspond to 9,764088866 10^{11} molecules. Since after forty cycles of PCR on 100 molecules we should have about $100\,2^{40} = 109.951.162.777.600$ molecules in 56 μl of solution (50 from the PCR mix plus 6 of dye), we can easily compute how much solution (Y) is enough to upload to have 100 ng of DNA in the gel.

$$
\begin{aligned}
&109.951.162.777.600\ \text{molecules} : 56\mu l = 9,764088866\,10^{11}\ \text{molecules} : Y\\
&Y = 0,249\mu l
\end{aligned}
$$

We usually upload 15 μl of solution in each well of the gel.

6.2.2 Gel-electrophoresis

The gel-electrophoresis technique, that separates DNA molecules by their length, is simple, rapid to perform, and capable of resolving fragments of DNA that cannot be separated adequately by other procedures. Since the DNA is negatively charged, after being placed in a small well of a gel in an active electric field, it slowly moves towards the positive side. Larger (heavier) molecules move slower and smaller molecules move faster. More precisely, they migrate through gel matrices at rates that are inversely proportional to the logarithm (of basis 10) of their length. Larger molecules have greater frictional drag, they worm the way through the pores of the gel less efficiently than smaller molecules. After a while the electric field is stopped, and the portion of the gel that contains molecules with the desired length can be cut out of the gel, DNA purified, and then used in subsequent experiments. Besides, the location of DNA within the gel can be determined directly by staining with low concentrations of the fluorescent intercalating dye ethidium bromide, and bands

[3] In other words, a mole of one base weighs approximately 325 g.

containing DNA can be detected by direct examination of the gel in ultraviolet light. Figure 6.1 from [4] shows the tool performing electrophoresis and an example of electrophoresis output.

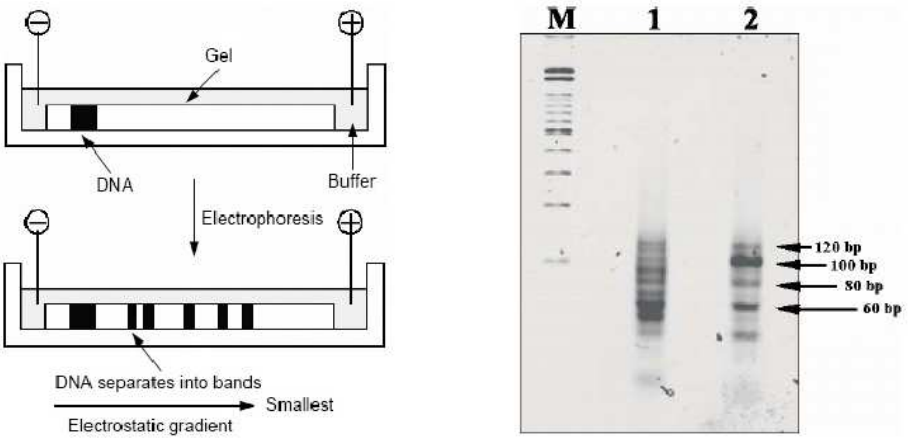


Fig. 6.1. Left side: Gel-electrophoresis process. Rigth side: Gel-electrophoresis results, where lane M is the DNA size marker, lanes 1 and 2 contain the tested molecules.

As first step of the procedure, the gel has to be prepared. Agarose and polyacrylamide are the most commonly used gels. We have chosen agarose gels, which are easier to prepare and to handle, have a lower resolving power but a greater range of separation than polyacrylamide gels. Moreover, the electrophoretic behaviour of DNA in agarose gels (in contrast to polyacrylamide gels) is not significantly affected by either the base composition of the DNA or the temperature at which the gel is run.

Agarose powder is extracted from seaweed, and is cast by melting it in the presence of the desired buffer[4] until a clear, totally transparent solution is achieved. The melted solution is then poured into a mold and allowed to harden, with a comb inserted along one side of the mold to form little wells in the gel. Upon hardening, the gel forms a matrix, the density of which is determined by the concentration of the agarose. An agarose gel has a concentration of $n\%$ whenever n grams of agarose are mixed with 100 ml of buffer. While n increasing from three on, during the melting in the microwave a lot of bubbles appear, that could interfere with the run of DNA or with the examination under ultraviolet light. We cannot wait that they go up while cooling down of the gel, because under 50°C it start to solidify,

[4] The electrophoretic mobility of DNA is affected by the composition and ionic strength of the electrophoresis buffer. In the absence of ions (if buffer is omitted from the gel by mistake), electrical conductance is minimal and DNA migrates very slowly, if at all. In buffer with high ionic strength (if a concentrated buffer is used by mistake) electrical conductance is very efficient and significant amounts of heat are generated (in the worst case, the gel melts and the DNA denatures). We have used a buffer containing EDTA, with pH 8.0, and Tris-acetate (TAE).

therefore such bubbles have to be taken away from the mold manually. Another thing to keep in mind is that any next time the gel is melted little part of the buffer will evaporate, and the concentration of the gel will increase. A linear DNA fragment of a given size migrates at different rates through gels containing different concentrations of agarose. There is a linear relationship between the logarithm of the electrophoretic mobility η of DNA and the gel concentration τ, which is described by the equation:

$$\log \eta = \log \eta_0 - K_r \tau$$

where η_0 is the free electrophoretic mobility of DNA and K_r is the retardation coefficient, a constant that is related to the properties of the gel and the size and shape of the migrating molecules. Approximately, we need a concentration n greater than 3 for lengths shorter than 100 ($n = 4$ for lengths 50-100 and $n = 5$ for lengths 20-50).

In order to detect the DNA bands, in the gel it is added 0.5 mg/ml of ethidium bromide[5], a fluorescent dye which intercalates between stacked base pairs. If the concentration is too high, then under the ultraviolet light the gel will result all over stained. Some loading buffer and colouring is instead added to the DNA solution respectively to make it heavier (in order to stay in the well) and visible during the run. One should be careful to the cases where sequences run as fast as the coloring, because the coloring could then cover the signal. A crucial point is to load a marker in a lateral well of the gel. A marker of 25bp for example[6] contains fragments of 25, 50, 75, 100 bp and so on until 200 bp. It is used as calibration path: different bands on this path mark known lengths, and the location of bands on other paths is compared with this calibration path (see rigth side of Figure 6.1)

At this point the gel is ready and the electric field can be switched on. We set 100 volt x 400 milliampere x 20 minutes. This choice is determined by experience (with such tools) and by the Ohm equation. We recall that in an electric field the electricity I is proportional to the difference of potential V by a coefficient depending on the means resistance R as

$$I = \frac{V}{R}$$

The point here is to set V high enough that the field remains uniform during the whole electrophoresis.

The system has to be stopped approximately when the sequences (visible by the coloring) reach an half of the maximum distance, by occupying an half of the gel container. In fact, if we stop before, the strings could be not yet separated, and if we stop after, the shortest strings would stay in a portion of gel without ethidium bromide (which run in the opposite direction of DNA!) and could not be detected by ultraviolet light. After we switch off the field we can see and photograph the signal. Imagines are obtained as those reported in the following of this chapter.

[5] It is a powerful mutagen, is carcinogen and should be handled with care!

[6] It could be a marker of 50bp or 10bp, respectively starting from strings long 50, or 100, and going ahead with differences of 25bp.

Coming back to our hypothetical experiment, we have just concluded the third step of the algorithm. What happens when PCR has not worked? Many and inexpected outcomes may be, included the electrophoresis result in Figure 6.2 that was managed to point out the signal. One can note how the coloring has run as sequences 50-70 long, and the excessive ethidium bromide has stained the gel.

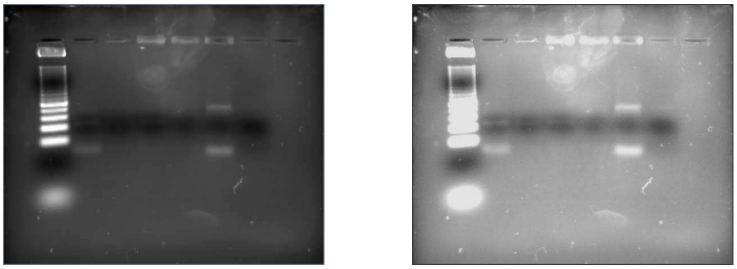


Fig. 6.2. This is a same picture of the "electrophoresis results" after a PCR, where the right one was manipulated by an image program to see better any signal. *Lane 1*: molecular size marker ladder (25b). *Lane 2*: Something 25 long seems to be present (primers?), *lane 3*: Nothing..? *lane 4, lane 5*: DNA seems to be still in the wells, *line 6*: one can see clearly sequences 20 long (primers) and a slight amplification 100 long

As notation for the next sections, we recall that a γ-superstrand is any strand which includes a substrand γ, and position $-n$ of a sequence indicates the position n in the backward direction. Often only primers of PCR are indicated when pools are clearly understood. Moreover, Latin capital letters are used to denote specific DNA sequences, that are written with respect to the usual $5' - 3'$ orientation.

6.3 XPCR-based extractions

In order to test the validity of XPCR, a first experiment was carried on with $\alpha\phi\gamma\psi$-strands, $\gamma\psi\beta$-strands, and primers α and $\bar{\beta}$ where $\alpha\phi\gamma\psi$ was a sequence extracted from RhoA, a human gene which regulates many essential cellular processes and controls cell activation in response to environmental cues (data are shown in Figure 6.3 which is taken from [72])[7].

[7] RhoA = ATGGCTGCC ATCCGGAAGA AACTGGTGAT TGTTGGTGAT GGAGCCTGTG GAAAGACATG CTTGCTCATA GTCTTCAGCA AGGACCAGTT CCCAGAGGTG TATGTGCCCA CAGTGTTTGA GAACTATGTG GCAGATATCG AGGTGGATGG AAAGCAGGTA GAGTTGGCTT TGTGGGACAC AGCTGGGCAG GAAGATTATG ATCGCCTGAG GCCCCTCTCC TACCCAGATA CCGATGTTAT ACTGATGTGT TTTTCCATCG ACAGCCCTGA TAGTTTAGAA AACATCCCAG AAAAGTGGAC CCCAGAAGTC AAGCATTTCT GTCCCAACGT GCCCATCATC CTGGTTGGGA ATAAGAAGGA TCTTCGGAAT GATGAGCACA CAAGGCGGGA GCTAGCCAAG ATGAAGCAGG AGCCGGTGAA ACCTGAAGAA GGCAGAGATA TGGCAAACAG GATTGGCGCT TTTGGGTACA TGGAGTGTTC AGCAAAGACC AAAGATGGAG TGAGAGAGGT TTTTGAAATG GCTACGAGAG CTGCTCTGCA AGCTAGACGT GGGAAGAAAA AATCTGGTTG CCTTGTCTTG TGA, α = RhoA[1,18]=ATGGCTGCCATCCGGAAG, γ = GAAGGATCTTCGGAATGATG at position -229 of RhoA, and $\overline{\beta}$ = GAACAGAAACTTATCTCAGAGGAA.

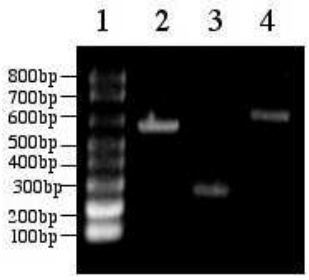


Fig. 6.3. Electrophoresis results (XPCR). *Lane 1*: molecular size marker ladder (100 bp). *Lane 2*: $\alpha\phi\gamma\psi$-strands of human RhoA (582bp); *lane 3*: $\gamma\psi\beta$-strands (253bp); *lane 4*: cross pairing amplification of $\alpha\phi\gamma\psi\beta$-strands (606bp): 606 = 582 +253 - 229.

Other experiments were performed with pools of sequences $\alpha\tau_2 \dots \tau_9\beta$ (150 long) generated by a combination of strands $X_2, X_3, \dots, X_9$, Y_3, Y_4, ..., Y_8, Z_2, Z_4, Z_6, Z_7, Z_9 [8] in such a way that τ_i is equal to X_i or Y_i or Z_i (for each i we have at least two choices). In particular, we started from a pool of eight different types of strands[9] where γ-superstrands $\alpha \dots \gamma$ and $\gamma \dots \beta\delta$ were obtained by standard PCRs and the expected results $\alpha \dots \gamma \dots \beta\delta$-strands were produced by XPCR procedure[10].

Finally, the complete algorithm was tested on a pool[11] in which γ' is present only in all the sequences that are not γ-superstrands[12] and all the γ-superstrands are either γ_1-superstrands or γ_2-superstrands[13]; all the steps were proved to be correctly performed (see lanes 2, 3, 4, 5 of Figure 6.4 which is taken from [72]), in fact all and only γ-superstrands were extracted from the pool. In order to verify the correctness of our results we performed three PCRs on the final pools. The first one with primers γ' and $\bar{\beta}$ showed that only γ-superstrands were present in the final pool, because there was no amplification of γ'-superstrands (see lane 6 of

[8] X_2 = CAAGATATGG, X_3= TCGTCTGCTAGCATG, X_4 = TCACGCCACGGAACG, X_5 = GTGAGCGCGAGTGTG, X_6 = ATATGCAATGATCTG, X_7 = ATCCGTCCCGATAAG, X_8 = CAAGTCAGATTGACC, X_9 = GCACGTAACT, Y_3 = CCCGATTAGTACAGC, Y_4 = TACTGATAAGTTCCG, Y_5 = TCGCTCCGACACCTA, Y_6 = TCAGCCGGCTTGCAC, Y_7 = AACTGATACGACTCG, Y_8 = TATTGTCACGCATCG, Z_2 = CAAGAGATGG, Z_4 = TCACGCCACGGAACT, Z_6 = TTAGCCGGCTTGCAC, Z_7 = TACTGATACGACTCG, Z_9 = GTACGTAACT, α = GCAGTCGAAGCTGTTGATGC, β = AGACGCTGCCGTAGTCGACG.

[9] $\alpha\ Z_2\ X_3\ X_4\ X_5\ X_6\ X_7\ Y_8\ Z_9\ \beta$, $\alpha\ X_2\ Y_3\ X_4\ X_5\ Z_6\ Y_7\ Y_8\ Z_9\ \beta$, $\alpha\ X_2\ Y_3\ X_4\ X_5\ X_6\ Z_7\ X_8\ X_9\ \beta$, $\alpha\ Z_2\ X_3\ Y_4\ Y_5\ Y_6\ Y_7\ X_8\ X_9\ \beta$, $\alpha\ Z_2\ X_3\ Z_4\ Y_5\ Y_6\ X_7\ Y_8\ Z_9\ \beta$, $\alpha\ X_2\ Y_3\ Y_4\ Y_5\ Y_6\ X_7\ X_8\ X_9\ \beta$, $\alpha\ Z_2\ X_3\ Y_4\ Y_5\ Y_6\ X_7\ Y_8\ Z_9\ \beta$, $\alpha\ X_2\ Y_3\ Y_4\ Y_5\ Y_6\ Y_7\ X_8\ X_9\ \beta$.

[10] γ = GAACGGTGAGCGCGAGTGTG in position 55 in all the strands where it occurs, and δ = CTTGTCTTTGAATAGAGTCTCCTT.

[11] $\alpha\ Z_2\ X_3\ X_4\ X_5\ X_6\ X_7\ Y_8\ Z_9\ \beta$, $\alpha\ X_2\ Y_3\ X_4\ X_5\ Z_6\ Y_7\ Y_8\ Z_9\ \beta$, $\alpha\ Z_2\ X_3\ Y_4\ Y_5\ Y_6\ Y_7\ X_8\ X_9\ \beta$, $\alpha\ X_2\ Y_3\ Y_4\ Y_5\ Y_6\ X_7\ X_8\ X_9\ \beta$, $\alpha\ X_2\ Y_3\ Y_4\ Y_5\ Y_6\ Y_7\ X_8\ X_9\ \beta$.

[12] $\gamma = Y_8$, $\gamma' = Y_4$.

[13] γ_1 = GATGGTCGTCTGCTAGCATG and γ_2 = TTAGCCGGCTTGCAAACTG.

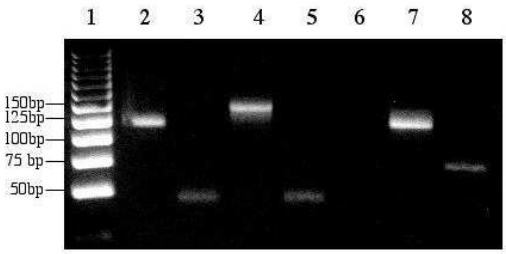


Fig. 6.4. Electrophoresis results (XPCR). *Lane 1*: molecular size marker ladder (25 bp). *Lane 2*: amplification of $\alpha \dots \gamma$ strands (120 bp); *lane 3*: amplification of $\gamma \dots \beta$ strands (45 bp); *lane 4*: cross pairing amplification of $\alpha \dots \gamma$ and $\gamma \dots \beta$ (150 bp). *Lane 5*: positive control by PCR$(\gamma, \bar{\beta})$, with γ at position -45; *lane 6*: negative control by PCR$(\gamma', \bar{\beta})$; *lane 7, 8*: positive controls by PCR$(\gamma_1, \bar{\beta})$ and PCR$(\gamma_2, \bar{\beta})$ respectively, with γ_1 at position -125 and γ_2 at position -75.

Figure 6.4). The last two PCRs with primers γ_1, $\bar{\beta}$, and γ_2, $\bar{\beta}$, respectively showed that all the initial γ-superstrands were present in the final tube (see lanes 7 and 8 of Figure 6.4).

6.4 XPCR-based generation

The following experiment showed that, given any SAT instance with 6 variables, the solution pool with sequences encoding all 64 assignments can be generated by starting from 4 specific sequences and by using only XPCR and electrophoresis. The success of this experiment is meaningful because it can be scaled up easily to any number n of variables: starting from 4 specific sequences, all 2^n combinations can be generated by XPCR as proved by Proposition 4.1.

The experiment started on an initial pool having the *four* sequences 150bp long[14]

$$\begin{aligned} S_1 &= \alpha\, W_a\, X_1\, X_2\, X_3\, X_4\, X_5\, X_6\, Z_a\, \beta \\ S_2 &= \alpha\, W_b\, Y_1\, Y_2\, Y_3\, Y_4\, Y_5\, Y_6\, Z_b\, \beta \\ S_3 &= \alpha\, W_a\, X_1\, Y_2\, X_3\, Y_4\, X_5\, Y_6\, Z_b\, \beta \end{aligned}$$

[14] S_1 = GCAGTCGAAGCTGTTGATGC CAAGAGATGG TCGTCTGCTAGCATG TCACGCCACGGAACG GTGAGCGCGAGTGTG ATATGCAATGATCTG ATCCGTCCCGATAAG CAAGTCAGATTGACC GCACGTAACT AGACGCTGCCGTAGTCGACG,
S_2 = GCAGTCGAAGCTGTTGATGC CAAGATATGG CCCGATTAGTACAGC TACTGATAAGTTCCG TCGCTCCGACACCTA TCAGCCGGCTTGCAC AACTGATACGACTCG TATTGTCACGCATCG GTACGTAACT AGACGCTGCCGTAGTCGACG,
S_3 = GCAGTCGAAGCTGTTGATGC CAAGAGATGG TCGTCTGCTAGCATG TACTGATAAGTTCCG GTGAGCGCGAGTGTG TCAGCCGGCTTGCAC ATCCGTCCCGATAAG TATTGTCACGCATCG GTACGTAACT AGACGCTGCCGTAGTCGACG,
S_4 = GCAGTCGAAGCTGTTGATGC CAAGATATGG CCCGATTAGTACAGC TCACGCCACGGAACG TCGCTCCGACACCTA ATATGCAATGATCTG AACTGATACGACTCG CAAGTCAGATTGACC GCACGTAACT AGACGCTGCCGTAGTCGACG.

$$S_4 \quad = \alpha\, W_b\, Y_1\, X_2\, Y_3\, X_4\, Y_5\, X_6\, Z_a\, \beta$$

where α and β, necessary to perform XPCR procedure, were 20b long[15], for $i = 1, 2, \ldots, 6$, X_i and Y_i, representing the boolean values of the variables of the problem, were 15b long[16], and the elongating sequences W_j and Z_j for $j = a, b$ were 10b long[17]. The steps of the experiment follow.

1. The initial pool was split randomly in four test tubes (a) (b) (c) (d), and the following PCRs were performed respectively
   a) $PCR(\alpha, \overline{X_2})$
   b) $PCR(X_2, \overline{\beta})$
   c) $PCR(\alpha, \overline{Y_2})$
   d) $PCR(Y_2, \overline{\beta})$
   so obtaining amplification of four types of sequences $\alpha \ldots X_2$, $X_2 \ldots \beta$, $\alpha \ldots Y_2$, and $Y_2 \ldots \beta$, that are 60, 105, 60, 105 long respectively (see lanes 2, 3, 4, 5 of Figure 6.5 which is taken from [77]).
2. Two $PCR(\alpha, \overline{\beta})$ were performed in parallel, one after having put together the product of (a) and (b), and the other one after having put together the product of (c) and (d) (sequences 150 long were amplified, see lanes 6 and 7 of Figure 6.5), then the two pools were mixed.

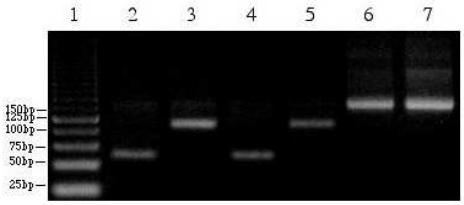


Fig. 6.5. Electrophoresis results (First step of XPCR-based generation). *Lane 1*: molecular size marker ladder (25bp). *Lane 2*: amplification of $\alpha \cdots X_2$ strands (60bp) and *lane 3*: amplification of $X_2 \cdots \beta$ strands (105bp), both PCRs performed at 52°C. *Lane 4*: amplification of $\alpha \cdots Y_2$ strands (60bp) and *lane 5*: amplification of $Y_2 \cdots \beta$ strands (105bp), both PCRs performed at 45°C. *Lane 6*: cross pairing amplification of $\alpha \cdots X_2$ and $X_2 \cdots \beta$ (150bp) and *lane 7*: cross pairing amplification of $\alpha \cdots Y_2$ and $Y_2 \cdots \beta$ (150bp), both XPCRs performed at 63°C.

3. For $i = 3$, 4, 5 the analogues of the previous steps (1) and (2) were performed by replacing index 2 with $i = 3, 4, 5$, and by referring to Figure 6.6 from [77], Top for $i = 3$, Middle for $i = 4$, and Bottom for $i = 5$ respectively.

[15] α = GCAGTCGAAGCTGTTGATGC and β = AGACGCTGCCGTAGTCGACG

[16] X_1 = TCGTCTGCTAGCATG, X_2 = TCACGCCACGGAACG, X_3 = GTGAGCGCGAGTGTG, X_4 = ATATGCAATGATCTG, X_5 = ATCCGTCCCGATAAG, X_6 = CAAGTCAGATTGACC. Y_1 = CCCGATTAGTACAGC, Y_2 = TACTGATAAGTTCCG, Y_3 = TCGCTCCGACACCTA, Y_4 = TCAGCCGGCTTGCAC, Y_5 = AACTGATACGACTCG, Y_6 = TATTGTCACGCATCG.

[17] W_a = CAAGAGATGG, W_b = CAAGATATGG, Z_a = GCACGTAACT, Z_b =GTACGTAACT

4. An electrophoresis was performed to select the sequences 150 long among longer ones (generated by unspecific amplification).

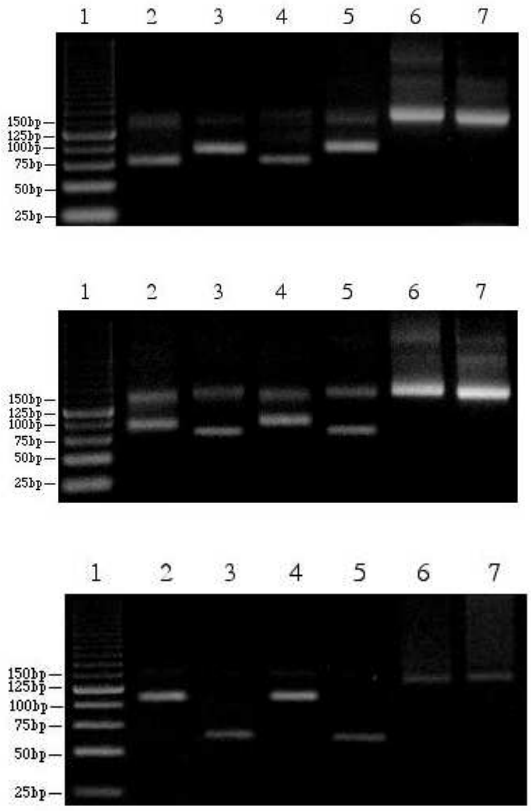


Fig. 6.6. Electrophoresis results (Final steps of XPCR-based generation). *Lane 1*: molecular size marker ladder (25bp). **Top.** *Lane 2*: amplification of $\alpha \cdots X_3$ strands (75bp), *lane 3*: amplification of $X_3 \cdots \beta$ strands (90bp), *lane 4*: amplification of $\alpha \cdots Y_3$ strands (75bp), *lane 5*: amplification of $Y_3 \cdots \beta$ strands (90bp), all PCRs performed at 52°C. *Lane 6*: cross pairing amplification of $\alpha \cdots X_3$ and $X_3 \cdots \beta$ (150bp) and *lane 7*: cross pairing amplification of $\alpha \cdots Y_3$ and $Y_3 \cdots \beta$ (150bp), both XPCRs performed at 63°C. **Middle.** *Lane 2*: amplification of $\alpha \cdots X_4$ strands (90bp), *lane 3*: amplification of $X_4 \cdots \beta$ strands (75bp), *lane 4*: amplification of $\alpha \cdots Y_4$ strands (90bp), *lane 5*: amplification of $Y_4 \cdots \beta$ strands (75bp), all PCRs performed at 42°C. *Lane 6*: cross pairing amplification of $\alpha \cdots X_4$ and $X_4 \cdots \beta$ (150bp) and *lane 7*: cross pairing amplification of $\alpha \cdots Y_4$ and $Y_4 \cdots \beta$ (150bp), both XPCRs performed at 63°C. **Bottom.** *Lane 2*: amplification of $\alpha \cdots X_5$ strands (105bp), *lane 3*: amplification of $X_5 \cdots \beta$ strands (60bp), *lane 4*: amplification of $\alpha \cdots Y_5$ strands (105bp), *lane 5*: amplification of $Y_5 \cdots \beta$ strands (60bp), all PCRs performed at 45°C. *Lane 6*: cross pairing amplification of $\alpha \cdots X_5$ and $X_5 \cdots \beta$ (150bp) and *lane 7*: cross pairing amplification of $\alpha \cdots Y_5$ and $Y_5 \cdots \beta$ (150bp), both XPCRs performed at 63°C.

By the electrophoresis results one can note that the sequences 150 long present in the pool were amplified (in linear or exponential manner) during each step. Although this phenomenon did not disturb the correctness of the 'computation', it caused noise and useless occupation of space in a test tube by increasing the unspecific matter, as one can see clearly in Figure 6.6. An improved experiment could be performed very easily by inserting intermediate electrophoresis steps to clean the signal from the noise caused by these amplifications. For example, electrophoresis separations after the cutting steps of each XPCR, could eliminate the strands 150 long, for decreasing the production of unspecific matter.

6.4.1 Recombination witnesses

The success of the experiment was tested in a first stage by the presence of all the (four) recombination witnesses (see Proposition 4.8), that has been guaranteed by the amplification of the following PCRs on the final pool (see lanes 2, 3, 4, 5 of Figure 6.7).

1. $PCR(X_1X_2Y_3, \overline{Y_4X_5X_6})$
2. $PCR(Y_1Y_2X_3, \overline{X_4Y_5Y_6})$
3. $PCR(Y_1X_2X_3, \overline{Y_4Y_5X_6})$
4. $PCR(X_1Y_2Y_3, \overline{X_4X_5Y_6})$

These PCRs were performed with primers 45 long, that are very specific but very long with respect to primers of a standard PCR (see Section 6.2.1). In this case the melting temperature had to be lower than usual, thus one could suspect that only proper substrings of these primers were annealed with the sequences present in the pool and that amplifications above could have been generated by sequences different from $W_1 = X_1X_2Y_3Y_4X_5X_6$, $W_2 = Y_1Y_2X_3X_4Y_5Y_6$, $T_1 = Y_1X_2X_3Y_4Y_5X_6$ and $T_2 = X_1Y_2Y_3X_4X_5Y_6$ (respectively). In order to verify that it was not the case, two negative controls were performed by PCRs with primers 45 long whose only one should not be present in the pool. This primer was a subsequence of βMyc and was extracted from the human gene RhoA.

1. $PCR(\alpha, \overline{\beta Myc})$
2. $PCR(X_1X_2Y_3, \overline{\beta Myc})$

In both these last PCRs no amplification was obtained, as expected (see lanes 6 and 7 of Figure 6.7). However, such proof was not definitively convincing and more checks seemed necessary.

In the meantime Theorem 4.9 came up, thus checking the presence of just a couple of recombination witnesses in the final pool became enough for the success of the experiment. Thereafter, the presence of W_1 and W_2 in the final pool was guaranteed by the amplification of the following PCRs[18].

For each sequence the presence of two extremal pieces 1,6 is checked by PCR, then, on the amplification product of this PCR, the presence of pieces 2,4 is

[18] Thanks to an anonymous referee of the 11th International Meeting on DNA Computing, June 6-9 2005, London, Ontario, for his/her interesting comments and for suggesting us to find a better procedure checking the complete recombination of the final pool.

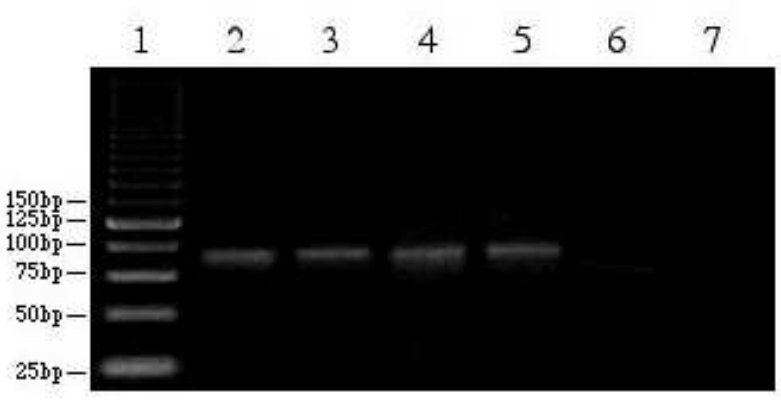


Fig. 6.7. Electrophoresis results (Recombination witnesses). *Lane 1*: molecular size marker ladder (25bp). *Lane 2*: positive control by $PCR(X_1X_2Y_3, \overline{Y_4X_5X_6})$ (90bp), *lane 3*: positive control by $PCR(Y_1Y_2X_3, \overline{X_4Y_5Y_6})$ (90bp), *lane 4*: positive control by $PCR(Y_1X_2X_3, \overline{Y_4Y_5X_6})$ (90bp), *lane 5*: positive control by $PCR(X_1Y_2Y_3, \overline{X_4X_5Y_6})$ (90bp), *lane 6*: negative control by $PCR(\alpha, \overline{\beta Myc})$, *lane 7*: negative control by $PCR(X_1X_2Y_3, \overline{\beta Myc})$, all PCRs performed at 63°C.

checked, and finally, the presence of pieces 3, 4 is checked on the product of the second PCR. A similar check with 'nested primers' was used in [108] as 'graduated PCR' to take the result of a computation. The nature of our problem allows us to implement an improvement of that technique, in fact a minor number of steps suffices.

We indicated with $Z_1Z_2Z_3Z_4Z_5Z_6$ a generic recombination witness, and for each of them the following procedure was executed in laboratory on a distinct copy of the pool P resulting from the experiment. That is, firstly the pool was split in two pools P_1 and P_2 with the same approximate size, and then on each of them the presence of a recombination witness $Z_1Z_2Z_3Z_4Z_5Z_6$ was checked by means the following steps:

1. **perform** $PCR(Z_1, \overline{Z_6})$
2. perform **electrophoresis** and select strands 90 long
3. **perform** $PCR(Z_2, \overline{Z_5})$
4. perform **electrophoresis** and select strands 60 long
5. **perform** $PCR(Z_3, \overline{Z_4})$

output: YES if the last PCR amplifies (sequences 30 long), NO otherwise.

The procedure proved the presence of $Z_1Z_2Z_3Z_4Z_5Z_6$ in the pool P because i) after the first two steps all and only the strands $Z_1 \cdots Z_6$ of P were present in the resulting pool (lines 2 and 5 of Figure 6.8), ii) after the second PCR and electrophoresis all and only the strands $Z_2 \cdots Z_5$ from strands $Z_1Z_2 \cdots Z_5Z_6$ of P were present in the resulting pool (lines 3 and 6 of Figure 6.8), iii) and the last

PCR amplified the portions Z_3Z_4 of such strands, that are found if and only if the sequence $Z_1Z_2X_3Z_4Z_5Z_6$ was present in P (lines 4 and 7 of Figure 6.8 which is taken from [77]).

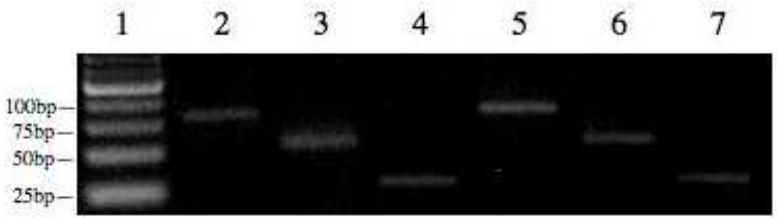


Fig. 6.8. Electrophoresis results (Checking the presence of the recombination witnesses). *Lane 1*: molecular size marker ladder (25bp). *Lane 2*: positive control by PCR $(X_1, \overline{X_6})$ (90bp) performed at 44°C. *Lane 3*: positive control by $PCR(X_2, \overline{X_5})$ (60bp) performed at 46°C. *Lane 4*: positive control by $PCR(Y_3, \overline{Y_4})$ (30bp) performed at 42°C. *Lane 5*: positive control by $PCR(Y_1, \overline{Y_6})$ (90bp) performed at 44°C. *Lane 6*: positive control by $PCR(Y_2, \overline{Y_5})$ (60pb) performed at 42°C. *Lane 7*: positive control by $PCR(X_3, \overline{X_4})$ (30pb) performed at 42°C.

6.5 XPCR-based mutagenesis

A pool of type $\{< \alpha\gamma\beta >\}$ was transformed in a pool of type $\{< \alpha\delta\beta >\}$ where γ is 123 bp long, δ is 150 bp long, and $\alpha\gamma\beta$ is the human gene RhoA[19], by the algorithm proposed in Section 5.4.

[19] α = ATGGCTGCCATCCGGAAG AAACTGGTGATTGTTGGT GATGGAGCCTGTG-GAAA GACATGCTTGCTCATAGT CTTCAGCAAGGACCAGTT CCCAGAGGTGTATGTGCC CACAGTGTTTGAGAACTAT GTGGCAGATATCGAGGTG GATGGAAAGCAGGTAGAG TTGGCTTTGTGGGACACA GCTGGGCAGGAAGATTAT GATCGCCTGAGGCCCCT CTCC-TACCCAGATAC,
γ = CGATGTTATACTGATGTG TTTTTCCATCGACAGCCC TGATAGTTTAGAAAACATC CCAGAAAAGTGGACCCCA GAAGTCAAGCATTTCTGTC CCAACGTGCCCATCATCC TGGTTGGGAATAA,
β = GAAGGATCTTCGGAATGATG AGCACACAAGGCGGGAGCT AGCCAAGAT-GAAGCAGGAG CCGGTGAAACCTGAAGAAG GCAGAGATATGGCAAACAG GATTG-GCGCTTTTGGGTACA TGGAGTGTTCAGCAAAGACC AAAGATGGAGTGAGAGAGGT TTTTGAAATGGCTACGAGAGC TGCTCTGCAAGCTAGACGTGG GAAGAAAAAATCTG-GTTGCCT TGTCTTGTGA,
δ = GCAGTCGAAGCTGTTGATGCCA AGAGATGGTCGTCTGCTAGCAT GTCACGC-CACGGAACGGTGAG CGCGAGTGTGATATGCAATGAT CTGATCCGTCCCGATAAGTATT GTCACGCATCGGTACGTAACTA GACGCTGCCGTAGTCGACG

PCR and XPCR results of XPCR-Mutagenesis algorithm are indicated in Figure 6.9[20].

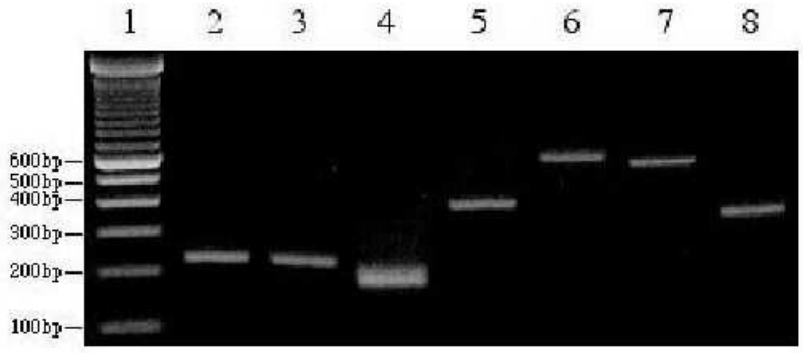


Fig. 6.9. Electrophoresis results (Mutagenesis). *Lane 1*: molecular size marker ladder (100bp). *Lane 2*: amplification of strand α (230bp). *Lane 3*: amplification of strand β (229bp). *Lane 4*: amplification of strand $\alpha[-18,-1]\delta\beta[1,20]$ (188bp). *Lane 5*: cross pairing amplification of α and $\alpha[-18,-1]\delta\beta[1,20]$ (400bp). *Lane 6*: cross pairing amplification of β and $\alpha\delta\beta[1,20]$ (609bp). *Lane 7*: RhoA (582bp). *Lane 8*: positive control by $PCR(\sigma,\beta[-20,-1])$ 354 bp. All PCRs are performed at temperature 58°C.

Experimental protocols

Reagents. 25 bp and 100 bp marker DNA ladder and agarose (Promega); PCR buffer, $MgCl_2$ and dNTPs (Roche); Taq DNA Polymerase (produced in laboratory); all the synthetic DNA oligonucleotides 150 bp long and all the primers were from Primm s.r.l.(Milano, Italy).

Annealing of synthetic DNA oligonucleotides. Two complementary synthetic 150 bp long DNA oligonucleotides ($5'-3'$ and $3'-5'$) were incubated at 1:1 molar ratio at 90°C for 4 min in presence of 2.5 mM of $MgCl_2$ and then at 70°C for 10 min. The annealed oligos were slowly cooled to 37°C, then further cooled to 4°C until needed.

PCR amplification. PCR amplification was performed on a PE Applied Biosystems GeneAmp PCR System 9700 (Perkin Elmer, Foster City, CA) in a 50 μl final reaction volume containing 1.25U of Taq DNA Polymerase, 1.5 mM $MgCl_2$,

[20] $\alpha[1,18]$ = ATGGCTGCCATCCGGAAG, $mir(\alpha[-18,-1])$ = GTATCTGGGTAGGAGAGG, $\alpha[-18,-1]$ = CCTCTCCTACCCAGATAC, $\beta[1,20]$ = GAAGGATCTTCGGAATGATG, $mir(\beta[1,20])$ = CATCATTCCGAAGATCCTTC, $mir(\beta[-20,-1])$ = TCACAAGACAAGGCAACCAG, $\alpha[-18,1]\delta[1,19]$ = CCTCTCCTACCCAGATACGCAGTCGAAGCTGTTGATG, $\delta[-17,-1]\beta[1,20]$ = CGCTGCCGTAGTCGACGGAAGGATCTTCGGAATGATG, $\sigma = \alpha[26,46]$ = GATGGTCGTCTGCTAGCATG.

200 μM each dNTP, PCR buffer, 80 ng DNA template, 0.5-1 μM of forward and reverse primers. The reaction mixture was preheated to 95°C for 5 min. (initial denaturing), termocycled 30 times: 95°C for 30 sec (denaturing), different temperatures (see figures) for 30 sec. (annealing), 72°C for 15 sec. (elongation); final extensions was performed at 72°C for 5/10 min.

Preparation and running of gels. Gels were prepared in 7×7 cm plastic gel cassettes with appropriate combs for well formation. Approximately 20 ml of 4% agarose solutions were poured into the cassettes and allowed to polymerize for 10 min. Agarose gels were put in the electrophoresis chamber and electrophoresis was carried out at 10 volt/cm^2, then the bands of the gels are detected by a gel scanner. The DNA bands (final PCR products) of interest were excised from the gel and the DNA was purified from the gel slices by Promega Kit (Wizard SV Gel and PCR Clean-Up System).

The experiments reported in this chapter may be found in the following articles.

- G. F., C. Giagulli, C. Laudanna, V. Manca, *DNA Extraction by Cross Pairing PCR*, C. Ferretti G. Mauri C. Zandron eds, Preliminary Proceedings of 10th International Meeting on DNA Computing: DNA 10, pp 193-201, University of Milano Bicocca, Milan, Italy, June 7-10 2004.

- G. F., C. Giagulli, C. Laudanna, V. Manca, *DNA Extraction by XPCR*, C. Ferretti G. Mauri C. Zandron eds, 10th International Workshop on DNA Computing, DNA 10, Milan, Italy, June 2004, Revised Selected Papers, LNCS 3384, Springer, pp 104-112, 2005.

- G. F., V. Manca, C. Giagulli, C. Laudanna, *DNA Recombination by XPCR*, Revised Selected Paper in A. Carbone, N.A. Pierce Eds: DNA11, June 6-9 2005, London, Ontario, Canada, LNCS 3892, pp 55-66, 2006.

7

A Model for DNA Self-Assembly

It is not knowledge, but the act of learning,
not possession, but the act of getting there,
which grants the greatest enjoyment.

Carl Friedrich Gauss[1]

The process in which substructures, driven by their selective affinity, are spontaneously self-ordered into superstructures, is now widely referred to as *self-assembly*. DNA can shape itself into many forms to achieve its purpose in life. The crystal structure of the junction between two of its forms provides insight into how DNA might accomplish some of these acrobatics [220]. Algorithms and information, fundamental to technological and biological organization, are also an essential aspect of DNA self-assembly [199]. The construction of molecular scale structures is one of the challenge at the core of an emerging discipline of Nanoscience, which is at a critical stage of development. The powerful molecular recognition system employed in DNA base pairing is used for constructing experimentally the self-assembly of three dimensional DNA structures (see Figure 7.1 comprising of DNA cube taken from [41] and DNA octahedron taken from [246]).

In nature, DNA appears as linear double stranded molecule (in eukaryotes) or in a circular form (mostly in viruses and bacteria, prokaryotes). Circular DNA molecules can be obtained also in a laboratory by joining (ligating) the ends of a linear DNA. They have been used in several models of DNA based computers and as building blocks for DNA knots [111]. If one imagines of unravelling the familiar helix (that is named B-DNA) and then twisting it up the other way around, then the result (that is named Z-DNA) is not so visually appealing, and not so stable, but it has some intriguing biology [220]. Left-handed Z-DNA exists in nature as a higher-energy and inherently unstable form of the double helix. In theory, any knot can be constructed in laboratory by using right-handed B-DNA for negative crossings and left-handed Z-DNA for positive crossings [111]. The Borromean rings displayed in Figure 1.1 were assembled by using B and Z DNA [157].

[1] Letter to Bolyai, 1808.

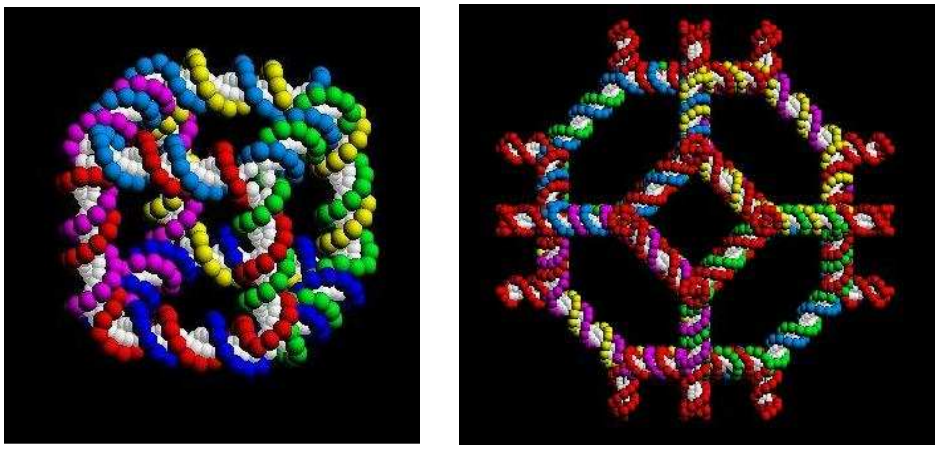


Fig. 7.1. DNA cube and octahedron were built up in laboratory.

Although there have been some notable successes in the self-assembly process, there are still lack of consistent methods for constructing complex structures out of pool of individual molecular components and in general, understanding the process of self-assembly remains challenging [218]. There are initial theoretical investigations dealing with complexity of the self-assembled structures and the computational power (see for example [116, 194, 222, 235]), however understanding how the molecular architecture works is a wide open question. This necessitates much more theoretical and experimental investigations.

In this chapter we present a theoretical model for the generation of DNA self-assembled forms as a family of graph structures that comply to certain "forbidden" constraints and follow some chemically predetermined "enforcing" properties [73]. This model is a variant of the forbidding-enforcing systems, that we present in the next section as the model of chemical processes originally introduced in [55].

7.1 Forbidding-enforcing systems

The forbidding-enforcing (f-e shortly) systems are a non-standard device to generate formal languages, alternative to the grammar systems from the classical formal language theory. This computational model was inspired by chemical processes and was used to simulate certain DNA based computations [55], and afterwards it was introduced in the context of the membrane computing [37].

The basic idea is to simulate a molecular reaction where "everything that is not forbidden is allowed". This assumes a completely different perspective with respect to the basic axiom underlying the computation by grammars and automata, where "everything that is not allowed is forbidden". In fact, while in a typical formal language theory model, a set of rewriting productions establishes how to generate (or recognize) words of a language, in a f-e system, a family of languages is generated by some *enforcing conditions*, dictating certain evolving rules for the system and some *forbidding conditions*, given as a group of patterns which cannot occur together at the same time. The enforcing rules ensure that if a certain

group of strings is present in the system, then some other strings will *eventually* be present as well.

More formally, given an alphabet Σ, we have the following definitions as introduced in [55].

Definition 7.1. *A forbidding set $\mathcal{F}$ is a family of finite nonempty subsets of Σ^+, and an enforcing set $\mathcal{E}$ is a family of ordered pairs (X, Y), where X and Y are finite subsets of Σ^+ and $Y \neq \emptyset$.*

We call a *forbidder* any element of $\mathcal{F}$ and an *enforcer* any element of $\mathcal{E}$.

Definition 7.2. *A forbidding-enforcing system (f-e system) is a triple $\Gamma = (\Sigma, \mathcal{F}, \mathcal{E})$, where $\mathcal{F}$ is a forbidding set and $\mathcal{E}$ is an enforcing set (over Σ).*

As usual, given a language L we denote with $sub(L)$ the set of all subwords of w for some $w \in L$.

Definition 7.3. *A language L over $\Sigma^\star$ is generated by an f-e system Γ if $F \not\subseteq sub(L)$ for every $F \in \mathcal{F}$, and $X \subseteq L \Rightarrow Y \cap L \neq \emptyset$ for every $(X, Y) \in \mathcal{E}$. The family of all languages generated by Γ is indicated by $\mathcal{L}(\Gamma)$ and is called f-e family.*

In order to generate a language, the evolution of an f-e system proceeds according to the "molecular reactions" specified by $\mathcal{E}$ (for every forbidder (X, Y), the presence of all the strings contained in X produces at least one of the strings contained in Y), *but* it is constrained by $\mathcal{F}$, that is, the evolution cannot lead to any group of patterns specified by a forbidder from $\mathcal{F}$. Note that the forbidding set $\mathcal{F}$ contains patterns, for example x and y, that *may not be* in the system, or that *may not be simultaneously* in the system, it depends on whether $\{x\}, \{y\} \in \mathcal{F}$ or $\{x, y\} \in \mathcal{F}$, respectively.

Definition 7.4. *An f-e system $\Gamma = (\Sigma, \mathcal{F}, \mathcal{E})$ is finitary if for any finite language Z there is a finite number of elements (Z, Y) in $\mathcal{E}$.*

In other words, a system is finitary if at any instance, the presence of a finite set of strings in the system enforces the presence of only a finite number of additional strings. Any family of languages that can be specified by an enforcing set can be also specified by a finitary enforcing set [55], thus there exists a sort of "finitary normal form" of f-e systems.

We conclude the introduction of the basic notions about f-e systems with a simple example. Let $\Sigma = \{a, b\}$, $\mathcal{F} = \{\{aa, bb\}\}$, and $\mathcal{E} = \{(\emptyset, bb)\}$. Then the subsets of $\Sigma^\star \backslash \Sigma^\star aa \Sigma^\star$ containing $\{bb\}$ make up the f-e family of the given system. To see this one can observe that the forbidder is satisfied by subsets from $\Sigma^\star \backslash \Sigma^\star aa \Sigma^\star$ or $\Sigma^\star \backslash \Sigma^\star bb \Sigma^\star$, since at least one of aa or bb cannot appear as a subword in the language. But, as the $\emptyset$ is a subset of any language, the enforcer ensures that bb is in every language of the family. Now $\Sigma^\star \backslash \Sigma^\star bb \Sigma^\star$ excludes bb, so all languages that satisfy the f-e system are subsets of $\Sigma^\star \backslash \Sigma^\star aa \Sigma^\star$ and contain $\{bb\}$.

In what follows we concentrate on annealing DNA strands guided by Watson-Crick complementarity. This will be performed by imposing forbidding-enforcing constraints guided by the physical and chemical constraints of the molecule. This

idea was inspired by the observation that in the forbidding-enforcing systems no element is transformed or destroyed during the evolution of the process, and elements are only added by the computation. If one wants to simulate DNA computations for example, more 'filtering' operations should be introduced in these systems, since the quantity of DNA strands should remain under a fixed threshold during the whole computation. But, if one wants to model the self-assembly process, where initial DNA filaments gradually are assembled in more complex structures while increasing the number of local interactions, this feature of forbidding-enforcing computation is just what one needs.

In fact, starting from a system containing some given DNA filaments that may be partially annealed, we model the formation of further bonds guided by Watson-Crick complementarity, without destroying or changing any of the bonds that are already present. Each DNA strand is represented as a directed path or a directed cycle (in the case of circular molecules) and the Watson-Crick connections are represented with undirected edges. This idea allows potentially larger number of "possible" products by the same DNA strands initially present in the pot. Therefore, instead of increasing the number of strings, we model the increase of annealing bonds among initial DNA substructures. In this way, we extend the basic idea of f-e systems to graphs and suggest another way to look into DNA self-assembly.

7.2 The model: graphs from paths

Regardless of the biochemical and topological properties of the structures seen in different aspects of DNA nanostructures, such forms can be seen as complex structures made of single strands connected (or attached respectively) to each other by two kinds of "bonds": phosphodiester bonds, i.e., concatenation and Watson-Crick complementarity.

We describe three-dimensional (3D) DNA forms by means of graphs $G = (V, \mathcal{P}, E, \xi)$ where V is a set of vertices labeled by elements of $\{a, c, g, t\}^k$ with k a fixed positive integer, $\mathcal{P}$ is a set of directed paths, possibly cycles, on the vertices of V, and E a partial matching set of undirected edges such that two vertices in V are incident with the same edge only if they have Watson-Crick complementary labels. The labelling function is $\xi : V \to \{a, c, g, t\}^k$. The labels of the paths (i.e., concatenation of the labels of the vertices) represent the given DNA filaments, while the edges of E represent the Watson-Crick bonds generated to form the structure. In Figure 7.2, which is taken from [74], a simple DNA form described by such graph is depicted.

A similar idea was used in [235] for describing a *DNA complex* on which self-assembly rules were defined. In that case, a DNA complex was considered as a connected directed graph with vertices labeled by symbols from $\{a, t, c, g\}$ and edges from $\{backbone, basepair\}$, with at most one incoming and one outgoing edge of each type at each node. Here we consider strings having a fixed length k as labels of the vertices, by abstracting the experimental fact that there exists a lower bound on these lengths that will provide Watson-Crick pairing. This bound depends on the temperature, salt concentration, and on other parameters of the

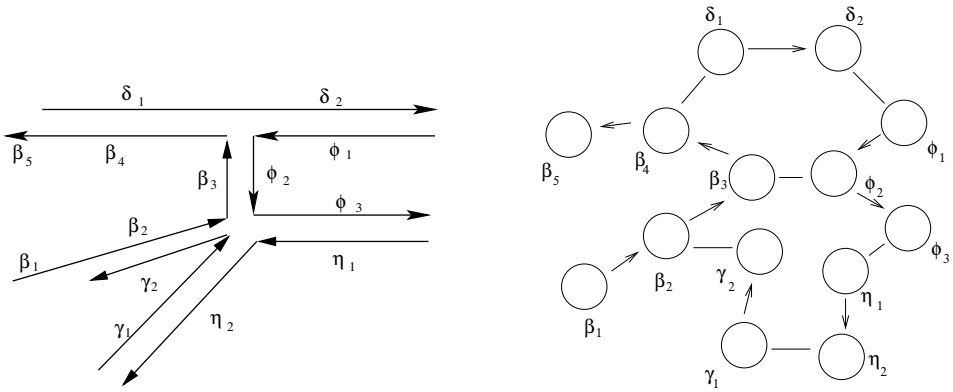


Fig. 7.2. Example of a self-assembled structure. One may note that: $\overline{\delta_1} = \beta_4$, $\overline{\delta_2} = \phi_1$, $\overline{\beta_3} = \phi_2$, $\overline{\eta_1} = \phi_3$, $\overline{\gamma_2} = \beta_2$ and $\overline{\eta_2} = \gamma_1$.

experiment. It is quite intuitive that starting from the same filaments the number of possible self-assembled DNA forms increases as the k value decreases, which in our case represents the (minimum) length of the attached portions. Here we focus on the graph structures corresponding to self-assembled forms, by fixing the value for k (for example, $k = 5$, which is approximatively the length of a half helical turn) and the complementarity between two strings of length k.

Another possibility that can appear experimentally but is ignored in this exposition is the overlapping of strings. Consider the ten symbol string $w = actactacta$. For $k = 5$, we can write $w = uv$ with $u = actac$ and $v = tacta$. However there are two occurrences of u as a substring of w, i.e., $w = act \cdot u \cdot ta$. In practice, a complementary string to u can anneal to both of these occurrences. Our model assumes that none of the strings representing DNA strands have such labelling.

On the other hand, we suppose that the correspondence from vertices to (labelling) strings may not necessarily be injective, in fact more than one occurrence of a string may be located along the filaments forming the structure. Therefore we consider a labelling function $\xi : V \to \{a, c, g, t\}^k$ that assigns a string from $\{a, t, c, g\}^k$ to each vertex from V. Further, in order to keep the model more realistic, all our graphs are *finite* graphs, where V and $\mathcal{P}$ are (given) finite sets.

The description of a self-assembly structure by means of such a graph simplifies the representation and emphasizes the interrelations between the substructures, for example a loop (a cycle with only one undirected edge) corresponds to a hairpin formation [209], and a connected component of the graph corresponds to one DNA structure.

Consider the triple cross over molecule TX [136] designed in the Ned Seeman's laboratory (see Figure 7.3 top). This complex structure is made of six strands, although there are examples of similar TX molecules with fewer strands. It can be presented as a graph in the following way. The length of the molecule is about 4.5 helical turns which corresponds to roughly 48 bases. For this, we consider 36 vertices, each labeled with a string of length 8. In Figure 7.3, which is taken from [73], (a) the TX molecule is simplified by ignoring the helical turns and the Watson-Crick pairing corresponding to 8 consecutive nucleotides is identified with

a group of five short bars. In Figure 7.3 (b) - (d) the process of obtaining a graph structure corresponding to the TX molecule is presented. The directed edges follow the $5'$ - $3'$ direction of the strand, and the undirected edges indicate the sequence of 8 base pairs.

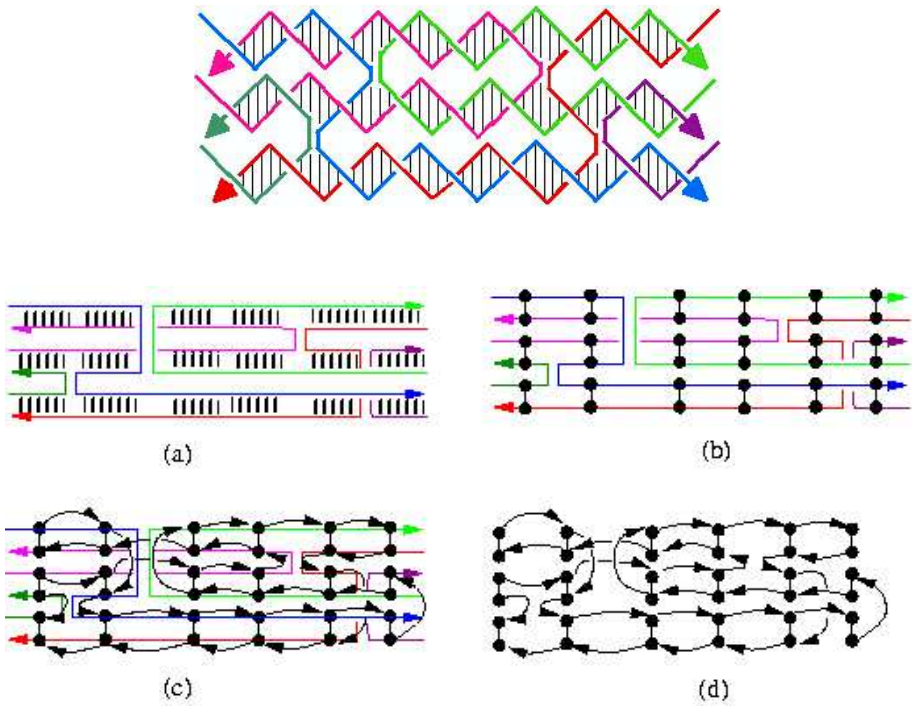


Fig. 7.3. Triple cross over molecule (TX) as a self-assembly graph.

If size is ignored, theoretically, from a few types of initial pieces, any shape can be self-assembled by spontaneous local bonding. In fact, the Kolmogorov complexity of a shape provides upper and lower bounds for the number of tile types necessary to self-assemble a given shape (at some scale) [222]. Nevertheless, to find strands that generate given structures remains a difficult design problem.

One can consider the converse question, what kind of forms are obtained by adding in a pot some given DNA substructures (or filaments). In other terms, assume a labeled graph structure $(V, \mathcal{P}, \xi, E)$ where V is the set of vertices, $\mathcal{P}$ the set of paths, ξ the labelling function, and E a given partial matching on the vertices. What are the edges that can be added in the (possibly empty) matching set E such that there exists a DNA structure corresponding to the obtained graph? It is clear that there exist graphs $(V, \mathcal{P}, \xi, E)$ that do not represent DNA structures, for example, they do not take in account some physical constraints. Figure 7.4, which is taken from [73], shows an example of two such structures.

Thus, given a collection of directed paths and cycles with vertices labeled by strings, we are going to consider (in Subsection 7.3.1) a set of *valid* graphs, where "forbidden" structures are not present. In particular, in order to obtain a graph

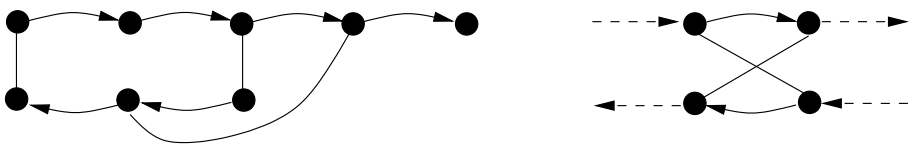


Fig. 7.4. Examples of graphs that do not correspond to a DNA structure.

which represents a self-assembled DNA structure, the matching set must respect certain constraints defined by means of a set of forbidden subgraphs which on the other hand follow physical and chemical restrictions of the DNA molecule. Such restrictions about the interrelations between DNA strings can be formulated only locally [119]. Moreover, a set of enforcing structures is considered in order to describe the parallelism intrinsic to the nature of self-assembly. This includes the considerations that further pairing of partially annealed molecules is preferred over strands that are far apart.

Molecular self-assembly is an inherently parallel process which begins anywhere it is energetically favored. Here we assume all thermodynamical conditions necessary for self-assembly are present such that assembly is obtained wherever it is structurally possible. Another assumption coming from the chemical structure of DNA is its *inflexibility*. For example, for a given k the model forbids formation of double stranded DNA circular molecules with length less than nk nucleotides. We can namely assume that $nk \geq 100$, in which case n would depend on our choice of k.

Let us consider in the next section the theoretical model for forbidding-enforcing systems on graphs [73].

7.3 Forbidding-enforcing graphs

Consider graphs of the type $G = (V, \mathcal{P}, E)$, where V is a finite set of vertices, $\mathcal{P}$ is a set of oriented paths, possibly cycles, on vertices of V, and E is partial matching, that is, a set of undirected edges such that any vertex of V is incident to at most one other vertex. We denote this family of graphs with $\mathcal{G}$. If $p \in \mathcal{P}$ we indicate with $A(p)$ the set of arcs included in the path p, and call $A(\mathcal{P})$ the set $\cup_{p\in\mathcal{P}} A(p)$.

Definition 7.5. *Given a positive integer m a m-local g-forbidder is a graph $(V, \mathcal{P}, E)$ in $\mathcal{G}$ with $|A(p)| < m$ for every $p \in \mathcal{P}$. A g-enforcer is an ordered pair (X, Y), where $X = (V, \mathcal{P}, E) \in \mathcal{G}$ and $Y = (V', \mathcal{P}', E') \in \mathcal{G}$ are such that $V = V'$, $\mathcal{P} = \mathcal{P}'$ and $E \subsetneq E'$.*

The constant m is included to ensure that all forbidders act locally, i.e., one only needs to concentrate on paths with not more than m vertices. This in general may depend on the experimental conditions. In what follows, we assume that m is fixed and all m-local g-forbidders are referred to simply as g-forbidders.

The set $\mathcal{F}$ of g-forbidders is called *forbidding set*, and the set $\mathcal{E}$ of g-enforcers is called the *enforcing set* of a family of graphs.

As in the original definition, a forbidding set may be infinite and the only requirement is that each forbidder is finite [55]. A g-forbidder is finite if it has a

finite number of arcs and edges. Moreover, regardless of the presence of the other forbidders, each g-forbidder cannot appear as subgraph of a graph satisfying that forbidder.

Definition 7.6. *A g-f-e system is a structure $\Gamma = (V, \mathcal{P}, E, \mathcal{F}, \mathcal{E})$, where V is a finite set of vertices, $\mathcal{P}$ is a set of directed paths, possibly cycles, on vertices of V, E is a partial matching on V, $\mathcal{F}$ is a set of g-forbidders and $\mathcal{E}$ is a set of g-enforcers.*

Given a graph $G = (V, \mathcal{P}, E) \in \mathcal{G}$, we call $sub(G)$ the set of all subgraphs $(V_0, \mathcal{P}_0, E_0)$ of G, where $V_0 \subseteq V$, every $p_0 \in \mathcal{P}_0$ is a path on vertices from V_0 such that $p_0 \subseteq p$ for some $p \in \mathcal{P}$ and $E_0 \subseteq E$ is a matching set on V_0. We write $G' \leq G$ for $G' \in sub(G)$. Similarly, $G' < G$ if $G' \leq G$ but $G' \neq G$.

Definition 7.7. *A graph $G = (V, \mathcal{P}, E^\star)$ is generated by the g-f-e system $\Gamma = (V, \mathcal{P}, E, \mathcal{F}, \mathcal{E})$ if, $E \subset E^\star$, $F \notin sub(G)$ for every $F \in \mathcal{F}$, and for every $(X, Y) \in \mathcal{E}$, if $X \in sub(G)$ then there is $Y' \in sub(G)$ such that $X < Y' \leq Y$.*

The family of all graphs generated by a graph-forbidding-enforcing system Γ is indicated by $\mathcal{G}(\Gamma)$. The elements of $\mathcal{G}(\Gamma)$ are called *assembled graphs*.

Similarly as in the original definition of forbidding-enforcing systems, the evolution of a g-f-e system proceeds according to the molecular reactions specified through $\mathcal{E}$ by increasing the elements of the matching set E, but not allowing subgraphs that are forbidden by $\mathcal{F}$.

7.3.1 Graphs for DNA structures

Now we concentrate on the model of g-f-e systems that simulate the self-assembly process of DNA. In this case the vertices of the graphs are labeled by strings from alphabet $\{a, g, c, t\}^k$. Hence, all graphs belong to the class of graphs $(V, \mathcal{P}, E, \xi)$ where V, $\mathcal{P}$, and E are the same as in the previous section, and $\xi : V \to \{a, g, c, t\}^k$ is the labelling of the vertices. All definitions for g-forbidders, g-enforcers and g-f-e systems are transfered to this class of graphs in a straight forward way. We note that the labelling of the vertices of every subgraph of a graph is preserved.

Consider a g-f-e system $\mathcal{G}(\Gamma)$ with $\Gamma = (V, \mathcal{P}, E, \xi, \mathcal{F}, \mathcal{E})$ where the DNA strings are associated to the paths $\mathcal{P}$ are given by the (finite number of) initial DNA filaments in the pot, the set E is given by the Watson-Crick bonds present in the initial DNA substructures. We specify a set of forbidders and enforcers that ensures construction of DNA structures.

The forbidding set $\mathcal{F}$ forbids constructions that are "impossible" by the physical and chemical properties of DNA. We list three g-forbidders that are most straightforward observations and should be included in every g-f-e system that simulates DNA self-assembly.

1. proper annealing (a pair of vertices is matched only if they have complementary labels)

$$F_0 = (V = \{v_1, v_2\}, \mathcal{P} = \emptyset, E = \{e = \{v_1, v_2\}\}, \xi(v_1) = \alpha, \xi(v_2) \neq \overline{\alpha})$$

2. hairpin constraint (a strand with a string $\alpha\bar{\alpha}$ without any distance between α and $\bar{\alpha}$ cannot form a hairpin)

$$F_1 = (V = \{v_1, v_2\}, \mathcal{P} = \{p = (v_1, v_2)\}, E = \{e = \{v_1, v_2\}\}, \xi(v_1) = \alpha, \xi(v_2) = \overline{\alpha})$$

To ease notation, we describe the forbidders just by listing the labels of the vertices in the paths and the set E. This assumes that all vertices appearing in the listed paths are distinct. Hence, the above forbidder is written $F_1 = (\{\alpha\bar{\alpha}\}, \{\{\alpha, \bar{\alpha}\}\})$.

3. non-crossing (orientation preserving constraint)

$$F_2 = (\{\alpha_1\alpha_2, \bar{\alpha}_1\bar{\alpha}_2\}, \{\{\alpha_1, \bar{\alpha}_1\}, \{\alpha_2, \bar{\alpha}_2\}\}).$$

The forbidder F_1 says that a positive length (between the attached portions) is necessary to allow a strand to turn back and attach to itself [209]. The forbidder F_2 avoids physically impossible situations as those ones in Figure 7.4. Note that both structures presented in Figure 7.4 are forbidden by F_2.

Due to the experimental conditions, the purpose of the design as well as the length of the labelling strings the complete set of g-forbidders for the self-assembly system may include additional structures, for example the one shown in Figure 7.5, which is taken from [73].

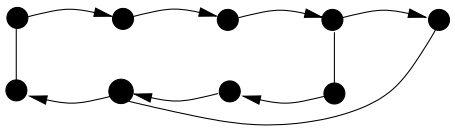


Fig. 7.5. The forbidden subgraph corresponding to some context constraint.

The basic set $\mathcal{E}$ contains the following g-enforcers. Arising from experimental evidence [239, 156, 199], it is clear that DNA strands prefer pairing with complete complements. Also, all of the DNA nanostructures are obtained by using "stick-end" cohesion. Hence, we have the following enforcers.

1. annealing $E_0 = (X, Y)$, where $X = (V = \{v_1, v_2\}, \emptyset, \emptyset, \xi(v_1) = \overline{\xi(v_2)})$ and $Y = (V = \{v_1, v_2\}, \emptyset, \{\{v_1, v_2\}\}, \xi(v_1) = \overline{\xi(v_2)})$.

   This enforcer can be seen as the brute-force enforcer that ensure annealing of the complementary edges. If left without any changes (say for example requiring that vertices v_1 and v_2 belong to paths with certain lengths) E_0 will ensure that all structures in the g-f-e system have all possible complementary vertices connected.

2. one side context rules, complete complements $E_1 = (X, Y)$ and $E_1' = (X', Y')$ where (by the simplified notation), $X = (\{\alpha\beta, \bar{\beta}\bar{\alpha}\}, \{\{\alpha, \bar{\alpha}\}\})$, $Y = (\{\alpha\beta, \bar{\beta}\bar{\alpha}\}, \big\{\{\alpha, \bar{\alpha}\}, \{\beta, \bar{\beta}\}\big\})$, $X' = (\{\alpha\beta, \bar{\beta}\bar{\alpha}\}, \big\{\{\beta, \bar{\beta}\}\big\})$, $Y' = (\{\alpha\beta,\ \bar{\beta}\bar{\alpha}\},\ \{\{\alpha, \bar{\alpha}\}, \{\beta, \bar{\beta}\}\})$.

E_1' is essentially the same as E_1 except that the initial hybridization has occurred at the other side of the molecule (one can see the two possibilities in Figure 7.6, which is taken from [74]). The enforcers E_1 and E_1' ensure that full complements of the strands are preferred.

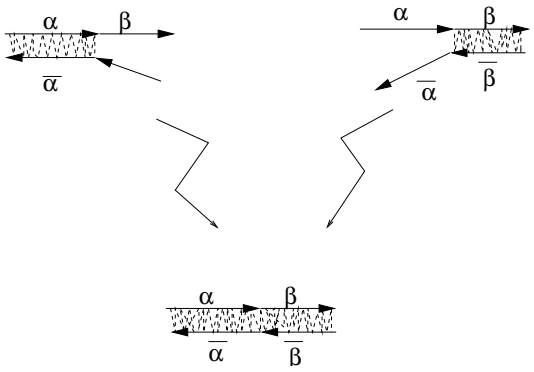


Fig. 7.6. One side context enforcing rule allowed by $\mathbf{E_1}$ and $\mathbf{E_1'}$.

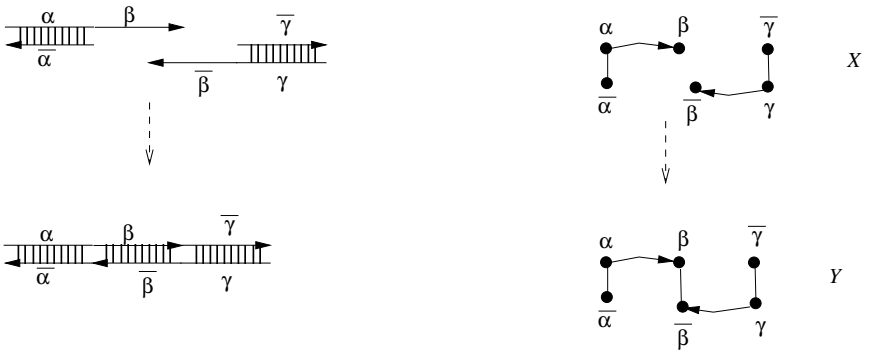
3. sticky end cohesion (see Figure 7.7 (a))
   $E_2 = (X, Y)$ where (by the simplified notation)
   $X = (\{\alpha\beta, \bar{\gamma}\bar{\beta}\}, \{\{\alpha, \bar{\alpha}\}, \{\gamma, \bar{\gamma}\}\})$
   $Y = (\{\alpha\beta, \bar{\gamma}\bar{\beta}\}, \big\{\{\alpha, \bar{\alpha}\}, \{\gamma, \bar{\gamma}\}, \{\beta, \bar{\beta}\}\big\})$

4. joining (see Figure 7.7 (b))
   $E_3 = (X, Y)$ where (by the simplified notation)
   $X = (\{\alpha\beta\gamma\delta, \bar{\delta}\bar{\gamma}, \bar{\beta}\bar{\alpha}\}, \emptyset)$
   $Y = (\{\alpha\beta\gamma\delta, \bar{\delta}\bar{\gamma}, \bar{\beta}\bar{\alpha}\}, \big\{\{\alpha, \bar{\alpha}\}, \{\beta, \bar{\beta}\}, \{\gamma, \bar{\gamma}\}, \{\delta, \bar{\delta}\}\big\})$

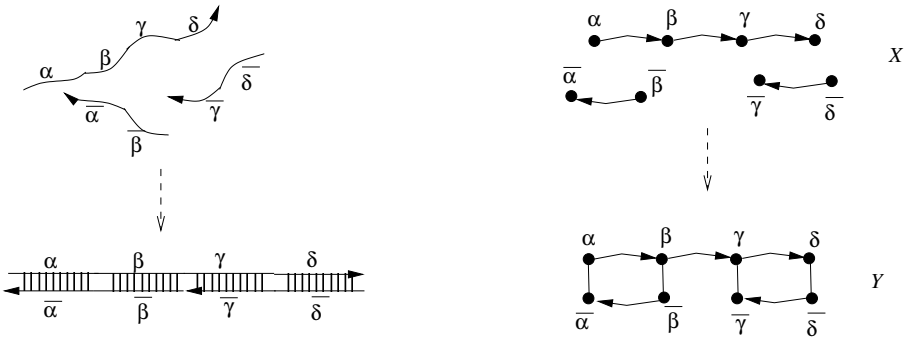
The enforcers that ensure sticky end cohesion are depicted in Figure 7.7, which was drawn by Nataša Jonoska (Mathematics Deptartment, USF, USA) for [73]. The left side of the figure is the partial annealing of the molecules and the right side depicts the graphs corresponding to these two cases. Note that the enforcer E_3 adds one, two, three or four new undirected edges. If only one edge is added, there is no guarantee that the full annealing will happen, but by the enforcers E_1 and E_1' there will be at least one more edge added.

As an example of enforced graphs we indicate the formation of the structured junctions as depicted in the Figure 7.8, which is taken from [74], where one can see some interactions between three, four, five and six initial paths. These formations are not present in nature, but they were all made in laboratory by self-assembly.

The sticky-end attachment simulated by the enforcer E_2 is basic for the formation of self-assembled structures (see Figure 7.8). The well known n-armed branched molecules, here enforced by E_3, were used to assembly polyhedrons, quadrilateral, the cube and the truncated octahedron in Figure 7.1 and seem particularly suitable for DNA graph constructions [118]. Figure 7.8 is quite abstract, in reality the angles between the "arms" are known to be flexible, at least if we



(a) enforcer $E_2 = (X, Y)$



(b) enforcer $E_3 = (X, Y)$

Fig. 7.7. The enforcing rules for sticky end cohesion (a) and "gluing" two molecules by a complement to both (b).

allow them to be 200 or 300 base pairs long, otherwise, short arms of two or three helical turns (one helical turn is about 10.5 bases) provides a rigid structure [111].

As in the case of forbidders, according to the experimental conditions and the initial designs of the molecules, additional enforcers may be added. However, we believe that the above set of enforcers should be included in every model of g-f-e system that describes DNA self-assembly.

We conjecture that the family $\mathcal{G}(\Gamma)$ defined by the g-f-e system described above exhibits all graph structures corresponding to DNA complexes that can be obtained from the initial substructures (given by $(V, \mathcal{P}, E, \xi)$) by means of self-assembly. Clearly it contains any DNA construction generated from the initial substructures, but it remains an open problem to show that the g-forbidders and the g-enforcers proposed above guarantee that this family contains *only* those graphs corresponding to possible DNA structures.

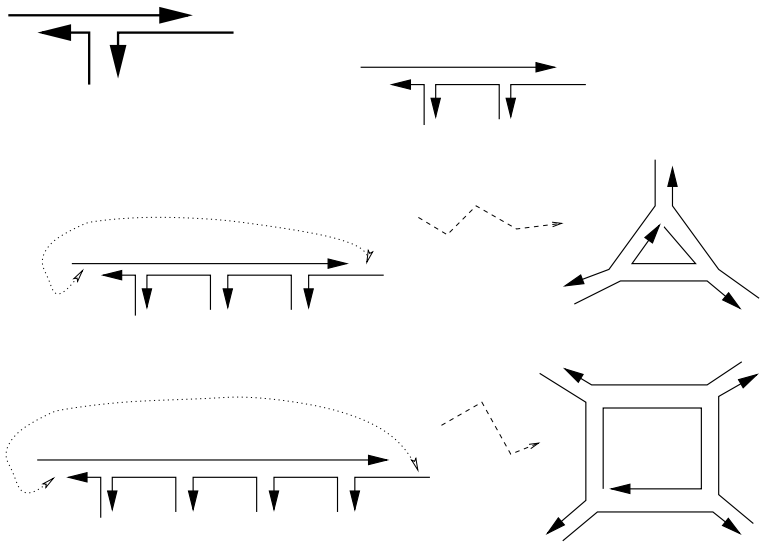


Fig. 7.8. Structured junctions starting from three, four, five, six paths ad hoc designed. The dot line indicates the sticky end rule allowed by E_2. Figure on the top left is a 3-junction. The other ones are structured junctions allowed by E_3.

Taking in account that the labels of the vertices are strings over a finite alphabet, one can consider theoretical questions in the context of formal language theory. It may be interesting to investigate the classes of graphs generated by a g-f-e system where the labels of V belong to a given language taken from one of the Chomsky classes. On the other hand, considering finite languages and investigating how the structure of generated graphs depends on the g-f-e system could be useful for the study of cellular processes, where for example the function of signal transduction nets is fairly well understood.

The content of this chapter was entirely published in the following articles.

- G. F., N. Jonoska, *Forbidding-Enforcing Conditions in DNA Self-Assembly*, J. Chen, N. Jonoska, G. Rozenberg eds: Nanotechnology, Science and Computation, pp 105 - 118, December 2005.

- G. F., N. Jonoska, *Forbidding-enforcing graphs for self-assembly*, 36th Southeastern International Conference on Combinatorics, Graph Theory, and Computing. Congressus Numerantium 177, pp 51 - 63, 2005.

8

Conclusions and Ongoing Research

The important thing is not to stop questioning.
Curiosity has its own reason for existing.

Albert Einstein[1]

The idea of unconventional computing has fired many imaginations, and many researchers regard it as a revolution in information processing [31]. In this respect, several directions are addressed in the first chapter of this thesis. An overview about molecular computing is given, in terms of state of the art and main problems faced, and conclusive notes are proposed along with bibliographical references and curiosities provoked by a few questions I was asked.

In Chapter 2 algorithmic features of DNA forms are presented from different viewpoints. The abstract and information-based logic underlying DNA structure seems to be paradigmatic in suggesting a general perspective of investigation, where computational aspects offer research inspirations both to biology and to computer science. Three string duplication algorithms are considered and compared. In particular, a 'bilinear duplication algorithm' is extended to a DNA-like computational system based on rewriting rules applied to double strings. This system is just a nice way to show how everything can be computed by suitably combining the actions of few enzymes on DNA double strings; but it could be a step forward, although on a theoretical level, in the purpose to manipulate natural processes to perform (universal) computations. On the other hand, the relative simplicity of the biological phenomena we use for this computation, points at the fact that most probably, natural phenomena are much more complex and may involve universal-like computations more often than what we suspect.

Molecular computations are then observed as transformations of objects inside and across the compartments of a membrane system. In this respect, a recently introduced strategy of rule application is recalled, that drops the idea of individual objects in favour of that of a dynamically controlled evolution of populations. In this context, the immunological process of selective leukocyte recruitment is briefly

[1] What I Believe, 1930.

described by a membrane system, and then simulated by a program implementing such a strategy of membrane rule application.

The middle chapters are essentially the core of the thesis. A variant of the Polymerase Chain Reaction, called XPCR, is introduced to implement null context splicing rules on DNA strings. This was experimentally tested in different situations, as implementation basis for algorithms of generation, extraction, and mutagenesis, and the laboratory experiments are reported in Chapter 6. The simple technology of this approach is interesting in itself, and it has different applications in biological contexts, beyond the DNA computing problems that have motivated it. It takes the advantages and the efficiency of an enzymatic elongation technique, and, it proves convenient with respect to the standard methods in terms of speed and feasibility.

XPCR-based algorithms turned out to be easy-to-implement methods supported by interesting mathematical facts. An analysis of the PCR process is presented in Chapter 3, where a novel extraction method based on XPCR procedure is proposed. This was successfully tested by the experiments described in Section 6.3. Its performance has been then improved, at least at a theoretical level, in [150], where a variant avoiding formations of chimers was introduced, together with the XPCR based mutagenesis algorithm that is described in Chapter 5.

In Chapter 4, we develop a XPCR-based method for implementing parallel single-point crossover, in order to generate combinatorial libraries of DNA strands, for DNA computation or other applications needing a similar library of DNA strands. The method proposed is not the first method for in vitro crossover to be proposed in DNA-based computing. Namely, parallel overlap assembly (POA) implements hybridization/polymerase extension-based crossover between homologous sequences: a comparison with our method is analyzed in Section 4.5.

A main difference is that, for a mixture of strands, and by employing the usual DNA polymerase, the XPCR method does not suffer of the competition which affects POA, because the recombination step of XPCR is defined in such a way that the two recombining portions have just a codeword in common. Therefore, hybridization between complementary regions (and polymerase extension) may correctly occur at only one point. Indeed, if mismatches occur at other points, this can be immediately detected because of a different (from the initial) length of product strings. Our method requires more steps than POA (which requires just 1 or 2, while our algorithm requires a linear number of steps) but it proves much more convenient from different points of view. Namely, we have less unwanted side products, and we can test the completeness of our library. This is a novelty with respect to all other generation methods. In this respect, combinatorial properties of strings from a pool produced by implementing the XPCR-based quaternary algorithm of generation have been proven in Sections 4.2 and 4.3. The algorithm and all the related combinatorial results have been generalized in Section 4.4 to the generation of non-binary libraries.

The algorithm of 'quaternary recombination' is implemented by a polymerase extension, which is a very efficient enzyme elongation method. This was tested by the experiment reported in Section 6.4, that showed the potential of the method. However, more experimental results are to be obtained to assess the scalability of the algorithm.

The theoretical design of an algorithm to solve an instance of SAT with significant dimension is described in Chapter 5. It follows the traditional, exhaustive search of solutions. Its generation procedure starts from only two strings and requires less steps than the quaternary recombination algorithm ($\log n$ versus n). Its extraction algorithm is implemented by a combination of polymerase extension and restriction endonuclease action (cutting operations). This algorithm may have a significant impact in the DNA computing area, because it potentially improves the existing methods to perform generating and extracting steps of the traditional extract model.

Finally, the model for self-assembly which is proposed in Chapter 7 suggests new directions both in graph theory and in DNA self-assembly. The general problem here faced is the following: given a set $\mathcal{P}$ of paths and cycles, a set $\mathcal{F}$ of forbidden structures, and a set $\mathcal{E}$ of enforced structures, what are the graphs included in the set $\mathcal{G}(\Gamma)$ for $\Gamma = (V, \mathcal{P}, E, \lambda, \mathcal{F}, \mathcal{E})$? We extend the basic idea of the forbidding enforcing systems, that were introduced as unconventional model of molecular processes in [55], to characterize graphs of (mathematical and biological) interest.

In particular, the presented model focuses on DNA self-assembly and the set of structures obtained through this process. However, the idea of graph forbidding enforcing systems can certainly be extended to other self-assembly processes in nature as well as in the pure theoretical methods to study mathematical properties of graphs. In the case of DNA self-assembly, the evolution process is described in a very natural way as an increase of the cardinality of the matching set between vertices with complementary labels. For other types of applications the concept of g-f-e systems may need to be adjusted in a different way, that would be more suitable to simulate the evolution in those particular processes.

Coming back to DNA computing, we point out that there has been so far little progress towards demonstrating a DNA computation that cannot be trivially performed in conventional machines. It is a very important open question to decide whether DNA computing will at some point reach the level of being useful from a practical point of view. On the other hand, the ultimate goal is not restricted to the study of realistic systems, implementable today or in the immediate future. The aim rather is to look for plausible computers and for models of them. In fact, progress in technology, whether physical or biochemical, is so rapid (and unpredictable) that it would be unwise to limit consideration to knowledge and tools that are available today (or tomorrow) [31].

My current research is essentially comprised of two main themes, which are described in the next sections along their very first results. One is that of looking for a good algorithm to silence genes, what we could call "DNA silencing" (collaboration with Vincenzo Manca, University of Verona). The other one is the modelling (by a membrane system) of a particular (unknown enough) phenomenon regarding the knee arthritis, in order to understand its basic dynamics (collaboration with Nataša Jonoska, USF). Besides, a paper with Vincenzo Manca, Cinzia Giagulli and Carlo Laudanna (University of Verona), has been recently completed [150], where a formal notation and 'uniformity principles' are introduced from which the XPCR-based methods follow.

8.1 DNA silencing

In Section 3.4 we faced the following problem. *Extracting from a pool P all the strings including a given substring γ*; that is, obtaining a pool composed by all the strings belonging to P such that contain γ as a substring:

$$\{\alpha\gamma\beta \mid \alpha\gamma\beta \in P\}.$$

We proposed the XPCR based extraction algorithm that amplifies all the γ-superstrands by means of XPCR_γ.

Let us consider now the following 'complementary' problem. *Given a string γ, extracting all strands from the pool P but γ*. In other words, silencing the γ-gene from a heterogeneous pool of genes, and obtaining a pool composed by all the strings belonging to P but γ:

$$P \setminus \{\gamma\}.$$

Without loss in generality, we can assume that the strings of P have the same length n. We are looking for a possibly efficient algorithm solving the above 'silencing problem', and for example one could think of the following three algorithmic solutions.

1. By elongation.

   Let γ_1 and γ_2 be the flanking strings of γ, meaning that $\gamma = \gamma_1\xi\gamma_2$ for some string ξ. Let δ be a string "sufficiently long", say 20bp. The γ-genes are can be detected by performing the following algorithm.
   a) $P := PCR(\gamma_1, \gamma_2\delta)(P)$
   b) $P := El_n(P)$
   In fact, with a certain number of PCR steps, in (a) all the γ-genes are elongated by means of a PCR with an external primer (see Section 3.1). Afterwards, they have length $n + |\delta|$, and they can be detected and eliminated from P by length selection (an electrophoresis). The idea of elongating all the γ-strings could be performed also by mutagenesis, by inserting a 'sufficiently long' string in some point of the sequence γ.

2. By complementarity.

   Let $\gamma = \gamma_1\xi\gamma_2$, with ξ a 'very short' string, and let ζ be a 'very long' string. The following algorithm detects γ-strings by complementarity, by means of 'bubble' annealing of them with longer strings (for details on such a kind of annealing see [235]), that makes them heavier and separable by electrophoresis. A schematic representation of the pool P after the first three steps is displayed in Figure 8.1.
   a) $P := P \cup \{\gamma_1\zeta\gamma_2\}$
   b) $P := H(P)$
   c) $P := C(P)$
   d) $P := El_n(P)$

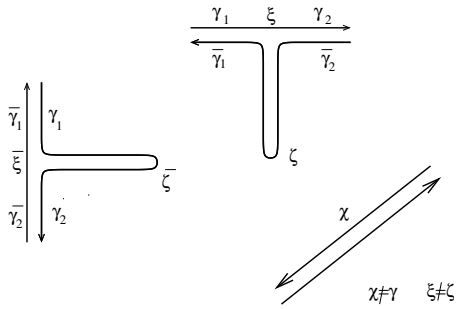


Fig. 8.1. Silencing by complementarity.

Selection by complementary is not very efficient though [1], and many γ-strings could remain unattached by the long ones.

3. By blocking PCR.

Let us suppose that all strands of the pool have been previously elongated by a prefix α and a suffix β. Note that they still have the same length $n + |\alpha| + |\beta|$. Let $\gamma = \alpha\gamma_1\xi\gamma_2\beta$, and two corresponding *blocking primers* $\eta_1 = \gamma_1\delta$, with $\delta \neq \xi$, and $\overline{\eta_2} = \overline{\gamma_2\zeta}$, with $\zeta \neq \xi$.
If we perform a special kind of PCR with four primers, two common to all the strings, and two blocking primers for γ, then all the strings of the pool are amplified but γ. In this case, the algorithm solving the silencing problem is just the operation:

$$P := PCR(\alpha, \eta_1, \eta_2, \beta)$$

meaning that a PCR with primers $\alpha, \eta_1, \overline{\eta_2}, \overline{\beta}$ is performed. The idea of blocking primers put at work is depicted in Figure 8.2. One may easily figure out that the above PCR amplify all the strings but those where η_1 and η_2 (only partially) attach, so blocking the polymerase primer extension.

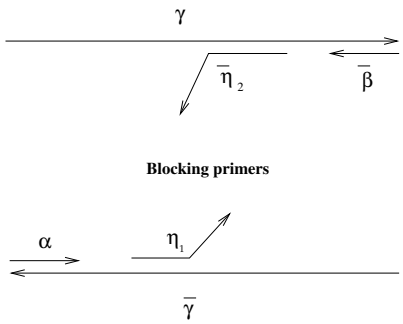


Fig. 8.2. Silencing by blocking.

First experiments have been carried on (during the writing of this thesis) to test the validity of blocking PCR, and quite encouraging results have showed up.

In Figure 8.3 there are the electrophoresis results of tests performed on five pools, four homogeneous (i.e., containing the only string $\alpha\gamma\beta$ to silence) called P_1, P_2, P_3, P_4, and one heterogeneous called P_t.

In the pool P_1 we had S_1 as template [2] and we exploited (η_1, η_2) as corresponding blocking primers[3], in P_2 we had S_2 as template [4] with (η_3, η_4) as corresponding blocking primers[5], in P_3 we had S_3 as template[6] having (η_5, η_6) as blocking primers[7], and in P_4 there was S_4 template[8] with (η_7, η_8) as blocking primers[9]. The pool P_t contained S_1, S_2, S_3, S_4, and the silencing was performed for all the sequences, by carrying on a blocking PCR with all the regular and blocking primers (see Figure 8.3). To increase the probability that blocking primers attach first than regular primers α and β[10], it is enough that they have a higher melting temperature (see Section 6.1).

Afterwards, we wondered if the silencing power of the blocking primers could depend on the primers concentration. Therefore, more experiments were carried out on P_t, again with all the blocking primers, but at different concentrations. It seemed to us that no substantial differences can be relieved (see Figure 8.4), thus the method would not depend significantly on primer (and blocking primer) concentrations.

8.2 On modelling knee injuries by membrane systems

We are interested to describe, in abstract and simplified terms, what happens in the knee tissue after an (internal) injury. It is not a well known phenomenon, and we focus some very recent experimental results that have stimulated our imagination and our interest. From the experiment described in Section 8.2.2 and carried on at the Medical School of University of South Florida, the interaction between a polymer called *hyaluronan* and its principal cell receptor seems quite relevant to understand some membrane modifications and the cell migration process. Also, at a molecular level, it is not known what is broken by an injury in the system. Therefore, as a starting point to model this phenomenon, in Section 8.2.3 we

[2] S_1 = GCAGTCGAAGCTGTTGATGCAAGAGATGGTCGTCTGCTAGCATGTCACGAATTAATTAATTAACCACGGAACGGTGAGCGCGAAGACGCTGCCGTAGTCG.

[3] η_1 = CAAGAGATGGTCGTCTGCTAGCATGTCACGCCCCCCCCCCCCCCCCC, $\overline{\eta_2}$ = TCGCGCTCACCGTTCCGTGGCCCCCCCCCCCCCCCCC.

[4] S_2 = GCAGTCGAAGCTGTTGATGCAATGATCTGATCCGTCCCGTTTAAATTTAAATTATAAGCAAGTCAGATTGACCGCACGTAACTAGACGCTGCCGTAGTCG.

[5] η_3 = CAATGATCTGATCCGTCCCGCCCCCCCCCCCCCCCCC, $\overline{\eta_4}$ = AGTTACGTGCGGTCAATCTGACTTGCTTATCCCCCCCCCCCCCCCCC.

[6] S_3 = GCAGTCGAAGCTGTTGATGCAAGATATGGCCCGATTAGTACAGCTACTGAAAATTTTAAAATTATAAGTTCCGTCGCTCCGACAGACGCTGCCGTAGTCG.

[7] η_5 = CAAGATATGGCCCGATTAGTACAGCTACTGCCCCCCCCCCCCCCCCC, $\overline{\eta_6}$ = GTCGGAGCGACGGAACTTATCCCCCCCCCCCCCCCCC.

[8] S_4 = GCAGTCGAAGCTGTTGATGACCTATCAGCCGGCTTGCACAAAAAAAAAAAAAAAACTCGTATTGTCACGCATCGGTACGTAACTAGACGCTGCCGTAGTCG.

[9] η_7 = ACCTATCAGCCGGCTTGCACCCCCCCCCCCCCCCCCC , $\overline{\eta_8}$ = AGTTACGTACCGATGCGTGACAATACGAGTCCCCCCCCCCCCCCCCC.

[10] α = GCAGTCGAAGCTGTTGATG, $\overline{\beta}$ = CGACTACGGCAGCGTCT.

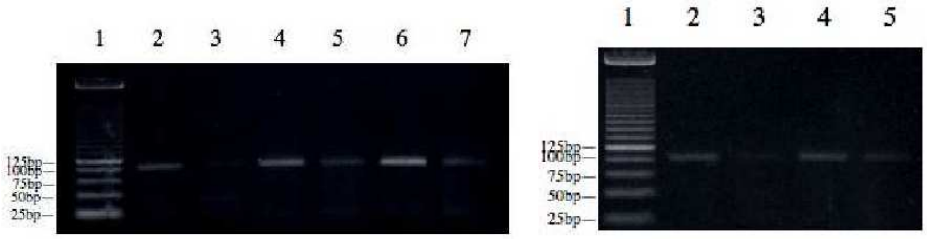


Fig. 8.3. Silencing by blocking. *Lane 1*: molecular size marker ladder (25bp). Left side. *Lane 2*: positive control on $P_1 = \{S_1\}$ by $PCR(\alpha, \beta)$, *lane 3*: negative control on $P_1 = \{S_1\}$ by $PCR(\alpha, \eta_1, \eta_2, \beta)$, *lane 4*: positive control on $P_2 = \{S_2\}$ by $PCR(\alpha, \beta)$, *lane 5*: negative control on $P_2 = \{S_2\}$ by $PCR(\alpha, \eta_3, \eta_4, \beta)$, *lane 6*: positive control on $P_3 = \{S_3\}$ by $PCR(\alpha, \beta)$, *lane 7*: negative control on $P_3 = \{S_3\}$ by $PCR(\alpha, \eta_5, \eta_6, \beta)$. Right side. *Lane 2*: positive control on $P_4 = \{S_4\}$ by $PCR(\alpha, \beta)$, *lane 3*: negative control on $P_4 = \{S_4\}$ by $PCR(\alpha, \eta_7, \eta_8, \beta)$. *Lane 4*: positive control on $P_t = \{S_1, S_2, S_3, S_4\}$ by $PCR(\alpha, \beta)$, *lane 5*: negative control on $P_t = \{S_1, S_2, S_3, S_4\}$ by $PCR(\alpha, \eta_1, \eta_3, \eta_5, \eta_7, \eta_2, \eta_4, \eta_6, \eta_8, \beta)$, that is a PCR with primers $\alpha, \eta_1, \eta_3, \eta_5, \overline{\eta_2}, \overline{\eta_4}, \overline{\eta_6}, \overline{\eta_8}, \overline{\beta}$. All PCR performed at $56°C$ on samples of $50\,\mu l$, with $0.25\,\mu l$ of each primer, at a standard concentration of $0.5\,\mu M$.

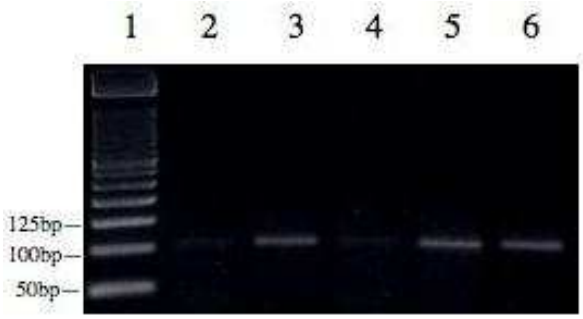


Fig. 8.4. Silencing by blocking - experimental variations. *Lane 1*: molecular size marker ladder (25bp). *Lane 2*: negative control on P_t by $PCR(\alpha, \eta_1, \eta_3, \eta_5, \eta_7, \eta_2, \eta_4, \eta_6, \eta_8, \beta)$ with $0.1\ \mu l$ of each primer, *Lane 3*: positive control on P_t by $PCR(\alpha, \beta)$ with $0.5\,\mu l$ of each primer, *Lane 4*: negative control on P_t by $PCR(\alpha, \eta_1, \eta_3, \eta_5, \eta_7, \eta_2, \eta_4, \eta_6, \eta_8, \beta)$ with $0.2\,\mu l$ of α and of $\overline{\beta}$ and $0.1\,\mu l$ of each blocking primer, *lane 5*: positive control on P_t by $PCR(\alpha, \beta)$ with $0.6\,\mu l$ of each primer, *lane 6*: positive control on P_t by $PCR(\alpha, \beta)$ with $0.1\ \mu l$ of each primer. All PCRs performed on samples of $50\,\mu l$ at $56°C$.

introduce a membrane system describing a system dynamics at a level of molecular interactions. As a next step, we intend to simulate the cell interactions and cell migration by extending concepts of membrane computing from [75, 115, 21]. A completely new feature of such a system would deal with the modifications of the cell membranes observed during the experiment (see the beautiful Figure 8.5, which was courtesy of Prof. Anna Plaas, from the Medical School of the University of South Florida, USA, together with the Figures 8.8 and 8.9).

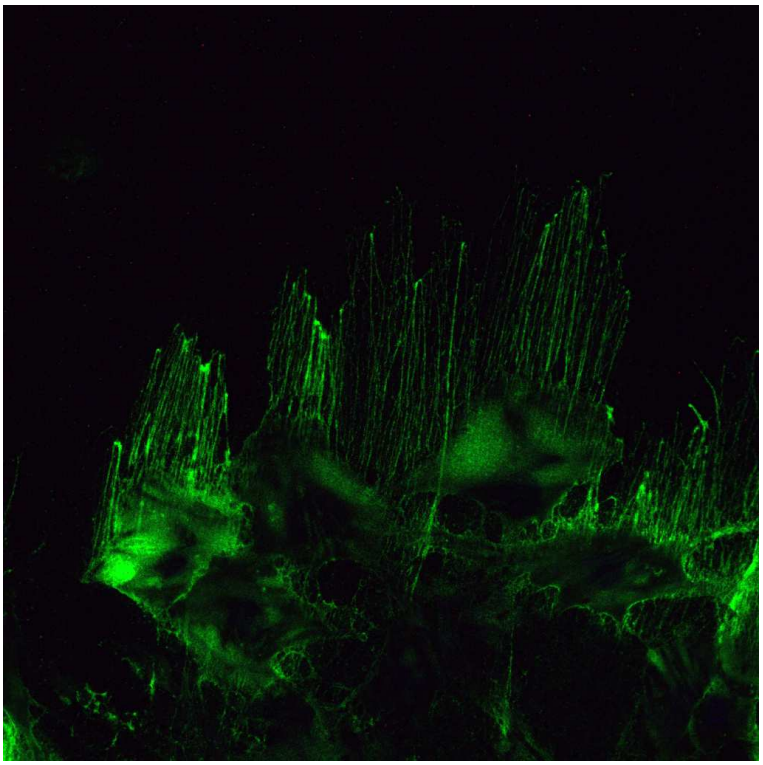


Fig. 8.5. Effects on cell membranes experimentally relieved after an injury.

8.2.1 The process

Firstly we present the scenario of the knee tissue in the healthy state (before the injury), so that we can then annotate the changes after an injury.

A tissue is like a community, where one can observe different characters, each with different behaviours. Here we can distinguish mainly four kinds of cells: *cartilage cells, synovial cells, stem cells* and *macrophages*. The last ones are all around in the tissue, generally passive and waiting that some infection is present to exhibit the specific receptors: more bacteria of an infection are present and more specific receptors are produced on the macrophages' surface to bind them, and then phagocyte them. The cartilage, synovial and stem cells are located as in Figure 8.6, where the macrophages are indicated by circles.

On the left side of Figure 8.6, with few more details in Figure 8.7, one may see the cartilage tissue. The cells called A have no particular activity, while those ones (called B) on the border can transform into macrophages and transmit molecules as signals for the cells nearby. Moreover, they proliferate to renew the part on the borderline that is continuously removed (by means of the fluid) by the articulation movement.

The fluid between the cartilage cells and the tissue on the other side allows the movement in the articulation. Its viscosity is mainly ruled by the quantity of hyaluronan polymer produced by the synovial cells. The functionality of such a

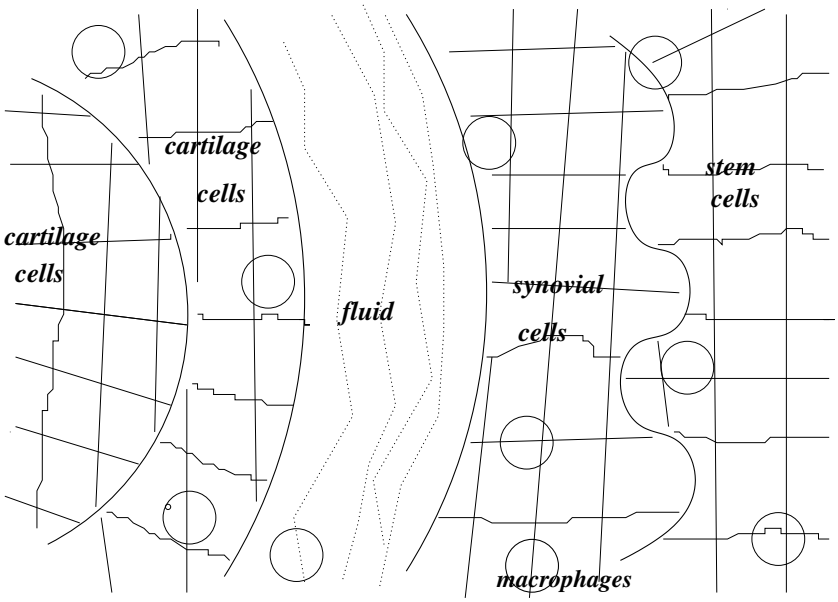


Fig. 8.6. The knee tissue nearby joints in a healthy state.

viscosity is controversial, especially for its role in tumor progression and metastasis, but in this context we can assume that in the healthy state we have an equilibrated quantity of hyaluronan, enough to hydrate and to feed the neighboring cells and to take away their waste products.

The most intriguing part of this zoom is on the right side of Figure 8.7, where synovial and stem cells are located. In a healthy state the stem cells just perform their turn-over: slightly proliferating and producing collagen, they die by apoptosis (gradual death). The synovium is a thin, weak layer of tissue only a few cells thick which lines the joint space. It acts to control the environment within the joint. It does this in two ways: first, it acts as a membrane to determine what can pass into the joint space and what stays outside; second, the cells within the synovium secrete substances, such as hyaluronan, which are the components of joint fluid.

After an injury, proliferation rates increase and new processes of cell differentiation and migration are activated by molecular signalling.

We can consider the total number of cells involved in this phenomenon unlimited, because they all can proliferate, and generally *the cells are not terminally differentiated*. In particular, a stem cell can transform into a synovial cell and then into a cartilage cell. The differentiation of cells happens mainly along the direction right-left in Figure 8.7, because the ultimate goal of the process is the reparation of the knee articulation. Cell differentiation and migration can be considered simultaneous, in fact the areas indicated in Figures 8.6 and 8.7 remain approximately the same during the whole process. For this reason we will identify such areas with regions of membranes.

New cartilage cells, forming the *scar tissue*, are thus produced after an injury and they are finally located all around the old cartilage cells (a similar phase

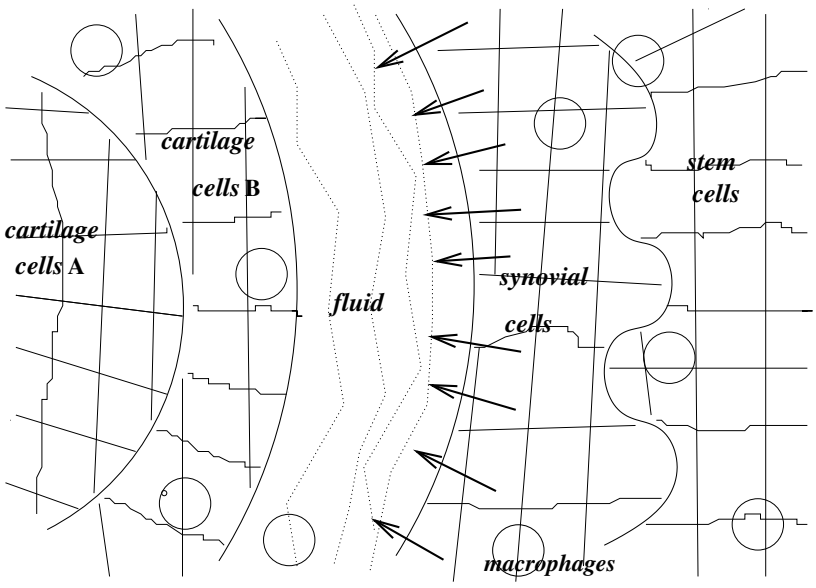


Fig. 8.7. Synovial cells secrete hyaluronan into the fluid.

can be clearly visualized in the skin reparation, see Figure 8.8) where they have arrived by migration. It seems to exist a limit though, to the number of cells that can differentiate to produce scar tissue: in fact sometimes the tissue cannot be completely repaired and arthritis effects are primed.

In this respect the role of the macrophages is surprising. In fact, they not only defend the environment from the external attacks of bacteria, but they are also able to send key signals that rule the behaviour of the other cells. Here we mention two of this substances: IL6 (interleukin-6, it is a chemokine [75]) and LIF (leukemia inhibitory factor). They are produced after an injury, the first one stimulates the cell differentiation, and the second one most likely stops the cell proliferation (some kind of control is necessary in order to avoid overproduction of cells). The production of LIF could be linked to the reason because the tissue sometimes is not completely repaired.

All these behaviours are primed and feed by signals, that are molecules released by cells and passing between cells. They propagate among cells nearby by means of physical contact. The propagation is cyclic: more active the cells are and more signals are produced. In terms of membrane systems the signals are objects, symbols or strings, while their quantity can be related to the length of strings or to the dynamics of the system.

8.2.2 Experiment

To understand the cell interaction of this phenomenon, a colony of cells of the same (synovial) type was isolated and an injury was simulated by injection of Transforming Grown Factor β (TGFβ). The cell reaction was the same than after an

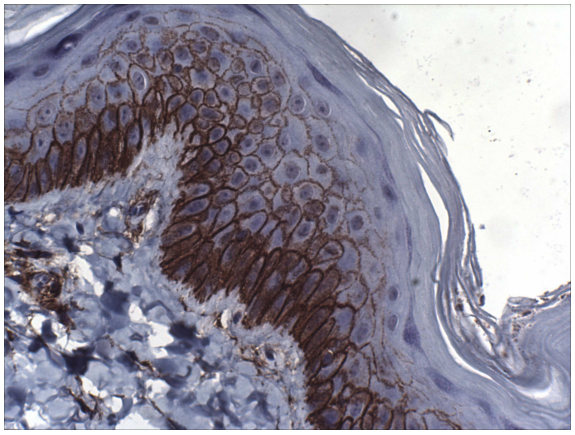


Fig. 8.8. Reparation by cell differentiation and migration in the bovine skin.

injury. The cells on the tissue surface around an hypothetic injury were stimulated to line, and the cells reaction was increased by injecting more and more TGFβ.

Some results were a wide production of hyaluronan polymer by the lining cells and the formation of membrane filaments directed towards the injury side (as in Figure 8.5). Therefore *the interruption of the interactions present in the healthy state destabilizes the regular form of the cell surface*, but the reason of that is as unknown as appealing. Recent studies[11] show that these formations are mainly stimulated and allowed by external environment factors, but they are also produced by the internal pushing of the enzyme producing hyaluronan. Moreover, no differentiation and two kinds of migration processes were relieved during the experiment, the planar one and the in-deep one, both towards the injury location.

The cellular movement induced by the presence of certain chemicals, towards those chemicals, is called *chemotaxis*. In the culture of the experiment no chemotactic locomotion occurred, because the cell migration happened towards the injury meanwhile the stimulating substance (TGFβ) was all over. But in the animal tissue, since the TGFβ is produced locally by cells reacting to the injury, a chemotactic migration likely happens. Finally, after the 'injury' the lining cells in the culture started to proliferate towards the empty space allowed by the injury. This reaction could be locally due just to the fact that they found room to proliferate.

An interesting fact to point out here is that the whole role of a cell in a tissue is dictated by external stimulates: what kind of cell it is, its functioning, and its behaviour. For example, all the cells have potential to make, bind, and interact with hyaluronan, but they do it or not depending on the environmental conditions.

In Figure 8.9 one can see the changing form of the cartilage tissue before (on the left), and after (on the right) the injury, when it needs to be repaired.

The green fluorescence relies the presence of hyaluronan, in fact, in the two figures an increasing production of hyaluronan can be observed. The red fluorescence

[11] Kirsi Rilla, Anne Kultti, Andrew Spicer, Raija Tammi and Markku Tammi, *Hyaluronan synthase and hyaluronan coat on MCF-7 breast cancer cells*, Hyaluronan oligosaccharide workshop, Boston, July 17, 2004.

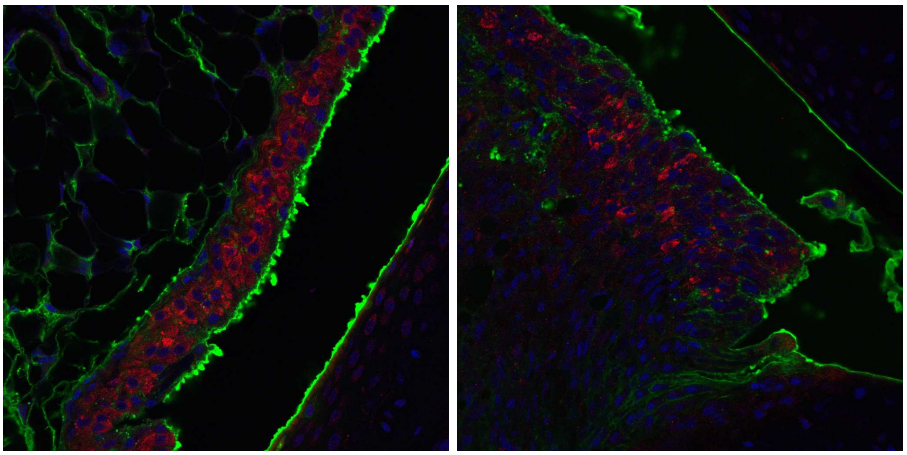


Fig. 8.9. Disposition of the synovial cells before and after the injury.

represents a different substance called *lubricin*. In a healthy state it is generally produced by the cartilage cells and retained by the lining (called B) cells, where we can observe an accumulation (see left part of Figure 8.9). After the injury something happens in such a way that the lubricin is spread out, maybe the production is interrupted, or maybe the lining cells release it all around.

In the next section, a membrane system is proposed as initial model. Technical details about the interaction between hyaluronan and its cell receptor would help to better understand the importance and the complexity underlying the whole phenomenon, but we are going slightly under the way.

8.2.3 A membrane system

The changes of the two main molecular productions observed during the experiment, regarding lubricin and hyaluronan, are here presented in a very schematic manner.

Let us identify the TGFβ with the symbol s (as *starting* the process after an injury), the hyaluronan with h, and the lubricin with l. Let us consider the two sides of an injury as nested membranes in which one can simulate the progressive action of the substance-symbol s over the cells in the most internal layers. The quantity of TGFβ is given by the length of the input string. The proper region of each membrane represents the layer of cells of a certain type, and the quantity of symbols from $\{a, b, c, d\}$ present in the initial configuration of the system (for example two in Figure 8.10) is proportional to the activity and production ability of the cells in the corresponding area. In a first approximation we have chosen three layers; the idea is to simulate the distribution of lubricin and hyaluronan in Figure 8.9 with the quantity of symbols l and h respectively.

The membrane M in the initial configuration reminds the presence of macrophage cells, which have a role in the molecular system. In fact, they produce LIF substance which stops the proliferation of the lining cells (corresponding to the area with symbols c), that is primed by the injury.

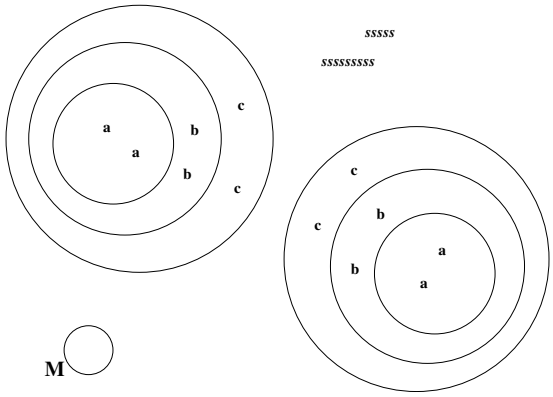


Fig. 8.10. Initial configuration of the membrane system simulating lubricin and hyaluronan production.

We propose the following global rules (i.e., unspecific for any region) on the alphabet $\{a, b, c, c', l, h, s\}$, that transform and exchange objects between membrane regions. We recall the semantic of the rules: the premises are transformed in the products by the rule wherever the premise symbols occur all together, while *here* and *in* indicate the region destination for objects introduced by the evolution rules. In particular, (ξ, in) means that ξ will enter any of the adjacent lower region (if there is any), chosen nondeterministically among all adjacent lower regions [180]. For example, after the application of the following first rule, a symbol s has entered one of the two bigger membranes of the system in Figure 8.10.

$$\begin{aligned} s\,s\,s\,s &\to (s\,s\,s,\ here)\ (s,\ in)\\ a &\to (a,\ here)\ (l,\ out)\\ b &\to (b,\ here)\ (l,\ out)\\ b\,l &\to (b,\ here)\ (l,\ out)\\ s\,s\,c\,l &\to (s\,s\,c\,c',\ here)\ (l,\ in)\\ s\,s\,a &\to s\,s\\ s\,s\,b &\to s\,s\\ c &\to c\,h\\ c' &\to (c'\,h,\ here)\ (h\,h,\ out) \end{aligned}$$

In this context the structure of string is not relevant, it just means the simultaneous presence of its elements. The application of the rules is nondeterministic and maximally parallel as usual; for example, if b, b, l are all present in a same region, then both the third and fourth rule are applied, and both the symbol l and an extra copy of it are sent out of that membrane.

The first rule simulates the progressive entry of the symbol s into the tissue and then into the most internal membranes. A layer can be passed through by the substance when it contains at least three copies of s (for example). The second, third, and fourth rules simulate the production of lubricin from the most internal layers, and its accumulation into the external one of the tissue, as happens during the healthy state (left part of Figure 8.9). The fifth rule simulates the release of lubricin by the lining cells after an injury, while the sixth and seventh rules

represent the slowing down of lubricin production. The last two rules simulate respectively the regular and then increasing production of hyaluronan after an injury.

This system is in a very infantile stage, it still needs to be tested and improved. Moreover, while setting a good molecular system, we should take in account that molecular productions are related to proliferation and differentiation processes. In fact, after an injury, the lining cartilage cells start to proliferate (as a consequence the molecular production of these cells increase tremendously) and to differentiate in macrophages, meaning that the number of cells sending signals to stop proliferation and to stimulate differentiation increases as well.

We wonder if the molecular production happening during cell proliferation and differentiation can be simulated only by evolution rules of objects. If a cell generates two cells producing a copy of the substance p, the quantity of p after n steps is $2^0 + 2^1 + \cdots + 2^n = 2^{n+1} - 1$. Moreover, these productions are "auto-ruled", as differentiation can occur, and differentiation in macrophages for example increases the production of substances that stop the proliferation.

Next goals for this modelling work are *i)* to define general rules that simulate the behaviour of healthy and injured tissue at the same time, *ii)* once the rules are established to identify some computational behaviour of the system (such as production of languages), *iii)* to identify good experimental conditions that will help to learn more precise rules of the system.

As final remarks, we point out the following observations related to this work with respect the membrane computing literature. The region of a membrane here represents a layer of cells of a certain type. In order to describe some phenomena, the differentiation and the movement (migration) of membranes should be allowed in a membrane system (a recent work in this direction is [17]). The cell membranes can change their form! In fact, as an experimental result, the interruption of the interactions present in the healthy state destabilizes the regular form of the cell surface (this is a consequence of the receptor-receptor interaction, a concept already formalized in modelling biological processes by membrane systems [75] and studied from a computational perspective [17]).

As a future research, I would like to carry on more experiments testing the scalability of the XPCR-based (extraction, generation, mutagenesis) methods introduced in Chapters 3, 4, and 6. An implementation in laboratory of the algorithm solving SAT proposed in Chapter 5 would be pretty interesting as well. In general, I'm interested in designing models and algorithms involving biological strings interactions—namely by investigating possible applications of the notion of DNA blocking introduced in Chapter 8 or by extending the self-assembly model presented in Chapter 7. Finally, as a modelling work, I intend to develop the model for reparation of knee injuries[12] presented in Chapter 8, and to continue the work [75] with (my advisor) Vincenzo Manca, by investigating immunological processes both from molecular (for example the genetic recombination of the ipervariable regions of antibodies) and cellular viewpoint.

[12] supported by the kind collaboration of Nataša Jonoska from Mathematics Department, Barbara Osborn and Anna Plaas from Medical School, of USF, USA.

References

1. L. M. Adleman. Molecular Computation of Solutions to Combinatorial Problems. *Science*, 266(5187):1021–1024, 1994.
2. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and J. D. Watson. *Molecular Biology of the Cell*. Garland Science, New York, fourth edition, 2002.
3. A. Alhazov, R. Freund, and Gh. Păun. Computational Completeness of P Systems with Active Membranes and Two Polarizations. In M. Margenstern, editor, *Machines, Computations, and Universality, International Conference, MCU 2004, Saint Petersburg, 2004, Revised Selected Papers*, volume 3354 of *Lecture Notes in Computer Science*, pages 82–92. Springer, 2005.
4. M. Amos. *DNA Computation*. PhD thesis, University of Warwick, UK, Sept 1997.
5. M. Amos. *Theoretical and Experimental DNA Computation*. Natural Computing Series. Springer, 2005.
6. A. Animashree, K. Krithivasan, and A. Choudhary. Breaking DES Using Peptide-Antibody Interactions. In A. Carbone, M. Daley, L. Kari, I. McQuillan, and N. Pierce, editors, *Pre-proceedings of DNA 11:. The 11th International Meeting on DNA Computing, June 6-9 2005, London, Ontario, Canada*, page 389, 2005.
7. I. Ardelean, D. Besozzi, M. H. Garzon, G. Mauri, and S. Roy. P System Models for Mechanosensitive Channels. In *[43]*, chapter 2, pages 43–82. Springer, 2006.
8. M. Arita. Writing Information into DNA. In *[117]*, pages 23–35, 2004.
9. P. Arrighi. Quantum Computation Explained to my Mother. *Bulletin of EATCS*, 80:134–142, June 2003.
10. B. Aspvall, M. F. Plass, and R. E. Tarjan. A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. *Information Processing Letters*, 8(3):121–123, 1979.
11. M. S. Balan, K. Krithivasan, and Y. Sivasubramanyam. Peptide Computing - Universality and Complexity. In *[120]*, pages 290–299, 2002.
12. N. Barbacari, A. Profir, and C. Zelinschi. Gene Regulatory Network Modelling by Means of Membrane Systems. In R. Freund, G. Lojka, M. Oswald, and Gh. Păun, editors, *Pre-proceedings of the 6th International Workshop on Membrane Computing, July 18-21 2005, Vienna, Austria*, pages 162–178, 2005.
13. E. B. Baum. Building an Associative Memory Vastly Larger than the Brain. *Science*, 268(5210):583–585, 1995.
14. Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro. Programmable and Autonomous Computing Machine made of Biomolecules. *Nature*, 414(1):430–434, 2001.
15. C. H. Bennett. Logical Reversibility of Computation. *IBM Journal of Research and Development*, 17:525–532, 1973.

16. C. H. Bennett. The Thermodynamics of Computation - a Review. *International Journal of Theoretical Physics*, 21(12):905–940, 1982.
17. F. Bernardini and M. Gheorghe. Cell Communication in Tissue P Systems: Universality Results. *Soft Computing*, 9(9):640–649, 2005.
18. F. Bernardini, M. Gheorghe, N. Krasnogor, R. C. Muniyandi, M. J. Pérez-Jiménez, and F. J. Romero-Campero. On P Systems as a Modelling Tool for Biological Systems. In *[81]*, pages 114–133, 2006.
19. F. Bernardini and V. Manca. P Systems with Boundary Rules. In Gh. Păun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Membrane Computing: International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002. Revised Papers*, volume 2597 of *Lecture Notes in Computer Science*, pages 107–118. Springer, 2002.
20. F. Bernardini and V. Manca. Dynamical Aspects of P Systems. *BioSystems*, 70(2):85–93, 2003.
21. L. Bianco, F. Fontana, G. Franco, and V. Manca. P Systems for Biological Dynamics. In *[43]*, chapter 3, pages 83–128. Springer, 2006.
22. L. Bianco, F. Fontana, and V. Manca. Metabolic Algorithm with Time-Varying Reaction Maps. In *Proceedings of the Third Brainstorming Week on Membrane Computing, Sevilla, Spain*, pages 43–61, February 2005.
23. L. Bianco, F. Fontana, and V. Manca. P Systems with Reaction Maps. *International Journal of Foundations of Computer Science*, 17(1):27–48, 2006.
24. L. Bianco and V. Manca. Metabolic Algorithms and Signal Transduction Dynamical Networks. In *First Brainstorming Workshop on Uncertainty in Membrane Computing, Palma de Mallorca, Spain*, pages 119–120, November 2004.
25. L. Bianco and V. Manca. Symbolic Generation and Representation of Complex Oscillations. *International Journal of Computer Mathematics*, to appear(2006).
26. D. Boneh, C. Dunworth, and R. Lipton. Breaking DES Using a Molecular Computer. In *[144]*, pages 37–65, 1995.
27. P. Bottoni, C. Martín-Vide, Gh. Păun, and G. Rozenberg. Membrane Systems with Promoters/Inhibitors. *Acta Informatica*, 38(10):695–720, 2002.
28. R. S. Braich, N. Chelyapov, C. Johnson, P. W. K. Rothemund, and L. M. Adleman. Solution of a 20-variable 3-SAT Problem on a DNA Computer. *Science*, 296(5567):499–502, 2002.
29. R. S. Braich, C. Johnson, P. W. K. Rothemund, D. Hwang, N. Chelyapov, and L. M. Adleman. Solution of a Satisfiability Problem on a Gel-Based DNA Computer. In *[44]*, pages 27–41, 2000.
30. K. J. Breslauer, R. Frank, H. Blocker, and L.A. Markey. Predicting DNA Duplex Stability from the Base Sequence. *Proceedings of the National Academy of Science of USA (PNAS)*, 83(11):3746–3750, 1986.
31. C. S. Calude and Gh. Păun. *Computing with Cells and Atoms – An Introduction to Quantum, DNA and Membrane Computing*. Taylor and Francis Inc, 2001.
32. A. Carbone and N.A. Pierce, editors. *Proceedings of the 11th International Workshop on DNA Computing: DNA 11, London, Ontario, Canada, June 2005. Revised Selected Papers*, volume 3892 of *Lecture Notes in Computer Science*. Springer, 2006.
33. A. Carbone and N.C. Seeman. Coding and Geometrical Shapes in Nanostructures: a Fractal DNA-assembly. *Natural Computing*, 2:133–151, 2003.
34. L. Cardelli. Bitonal Membrane Systems – Interactions of Biological Membranes. Microsoft Research, Cambridge, UK, October 2003.
35. L. Cardelli and A. D. Gordon. Mobile Ambients. *Theoretical Computer Science – Special Issue on Coordination*, 240(1):177–213, 2000.
36. G. Casiraghi, C. Ferretti, A. Gallini, and G. Mauri. A Membrane Computing System Mapped on an Asynchronous, Distributed Computational Environment. In *[81]*, pages 159–164, 2006.

37. M. Cavaliere and N. Jonoska. Forbidding and Enforcing in Membrane Computing. *Natural Computing*, 2(3):215–228, 2003.
38. M. Cavaliere, N. Jonoska, and P. Leupold. Recognizing DNA Splicing. In *[32]*, pages 12–26, 2006.
39. M. Cavaliere, N. Jonoska, S. Yogev, R. Piran, E. Keinan, and N. C. Seeman. Biomolecular Implementation of Computing Devices with Unbounded Memory. In *[64]*, pages 35–49, 2005.
40. J. Chen, R. Deaton, and Y.-Z. Wang. A DNA-Based Memory with *in vitro* Learning and Associative Recall. *Natural Computing*, 4(2):83–101, 2005.
41. J.H. Chen and N.C. Seeman. Synthesis from DNA of a Molecule with the Connectivity of a Cube. *Nature*, 350:631–633, 1991.
42. K. Chen and E. Winfree. Error Correction in DNA Computing: Misclassification and Strand Loss. In *[238]*, pages 49–62, 2000.
43. G. Ciobanu, Gh. Păun, and M. J. Pérez-Jiménez, editors. *Applications of Membrane Computing*. Natural Computing Series. Springer, 2006.
44. A. Condon and G. Rozenberg, editors. *DNA Computing: Proceedings of the 6th International Workshop on DNA-Based Computers, DNA 2000, Leiden, The Netherlands, June 13-17, 2000, Revised Papers*, volume 2054 of *Lecture Notes in Computer Science*. Springer, 2001.
45. M. Conrad. Information Processing in Molecular Systems. *Currents in Modern Biology*, 5(1):1–14, 1972.
46. M. Conrad. On Design Principles for a Molecular Computer. *Communication of the ACM*, 28:464–480, 1985.
47. E. Csuhaj-Varju, L. Kari, and Gh. Păun. Test Tube Distributed Systems Based on Splicing. *Computers and AI*, 15(2-3):211–232, 1996.
48. M. Daley and I. McQuillan. On Computational Properties of Template-Guided DNA Recombination. In *[32]*, pages 27–37, 2006.
49. J. Dassow and V. Mitrana. Splicing Grammar Systems. *Computers and AI*, 15(2-3):109–122, 1996.
50. R. Deaton, J. Chen, H. Bi, M. Garzon, H. Rubin, and D. H. Wood. A PCR-Based Protocol for In Vitro Selection of Non-Crosshybridizing Oligonucletides. In *[95]*, pages 196–204, 2002.
51. A. DeLuca and A. Restivo. A Characterization of Strictly Locally Testable and its Application to Subsemigroups of a Free Semigroup. *Information and Control*, 44(3):300–319, 1980.
52. R. DeSalle, M. Barcia, and C. Wray. PCR Jumping in Clones of 30-Million-Year-Old DNA Fragments from Amber Preserved Termites. *Experientia*, 49(10):906–909, 1993.
53. D. Dressman, H. Yan, G. Traverso, K. W. Kinzler, and B. Vogelstein. Transforming Single DNA Molecules into Fluorescent Magnetic Particles for Detection and Enumeration of Genetic Variations. *Proceedings of the National Academy of Science of USA (PNAS)*, 100(15):8817–8822, 2003.
54. A. Ehrenfeucht, T. Harju, I. Petre, D. M. Prescott, and G. Rozenberg. Formal Systems for Gene Assembly in Ciliates. *Theoretical Computer Science*, 292:199–219, 2003.
55. A. Ehrenfeucht, H. J. Hoogeboom, G. Rozenberg, and N. van Vugt. Forbidding and Enforcing. In *[238]*, pages 195–206, 2000.
56. A. Ehrenfeucht, I. Petre, D. M. Prescott, and G. Rozenberg. String and Graph Reduction Systems for Gene Assembly in Ciliates. *Mathematical Structures in Computer Science – Special Issue: Theory and Applications of Graph Transformations*, 12:113–134, 2002.

57. A. Ehrenfeucht, D. M. Prescott, and G. Rozenberg. Computational Aspects of Gene (un)scrambling in Ciliates. *Evolution as Computation*, pages 216–256, 2001.
58. A. Ehrenfeucht and G. Rozenberg. Forbidding-Enforcing Systems. *Theoretical Computer Science*, 292(3):611–638, 2003.
59. T. L. Eng. Linear DNA Self-Assembly with Hairpins Generate the Equivalent of Linear Context-Free Grammars. In *[204]*, pages 296–301, 1998.
60. T. L. Eng. On Solving 3CNF-satisfiability with an *in vivo* Algorithm. *[125]*, pages 135–141, 1999.
61. D. Faulhammer, A. R. Cukras, R. J. Lipton, and L. F. Landweber. Molecular Computation: RNA Solution to Chess Problems. *Proceedings of the National Academy of Science of USA (PNAS)*, 98(4):1385–1389, 2000.
62. D. Faulhammer, R. J. Lipton, and L. F. Landweber. Counting DNA: Estimating the Complexity of a Test Tube of DNA. *[125]*, pages 193–196, 1999.
63. C. Ferretti, S. Kobayashi, and T. Yokomori. DNA Splicing Systems and Post Systems. In L. Hunter and T. E. Klein, editors, *First Annual Pacific Conference on Biocomputing*, pages 288–299. World Scientific, 1996.
64. C. Ferretti, G. Mauri, and C. Zandron, editors. *Proceedings of the 10th International Workshop on DNA Computing: DNA 10, Milan, Italy, June 2004. Revised Selected Papers*, volume 3384 of *Lecture Notes in Computer Science*. Springer, 2005.
65. R. P. Feynman. Miniaturization. In D. H. Gilbert, editor, *Reinhold*, pages 282–296. New York, 1961.
66. F. Fontana, L. Bianco, and V. Manca. P Systems and the Modeling of Biochemical Oscillations. In *[81]*, pages 199–208, 2006.
67. F. Fontana and G. Franco. Finding the Maximum Element Using P Systems. *Journal of Universal Computer Science*, 10(5):567– 580, 2004.
68. G. Franco. Combinatorial Properties of DNA Libraries Generated via Null Context Splicing Rules. Submitted.
69. G. Franco. Algoritmi DNA per la Soddisfacibilità Proposizionale. Master's thesis, Dipartimento di Matematica, Università degli Studi di Pisa, November 2001.
70. G. Franco. Hard Instances of SAT. TR project 'Biomolecular Algorithms for NP-Complete roblems', Department of Computer Science, University of Pisa, July 2002.
71. G. Franco. A Polymerase Based Algorithm for SAT. In M. Coppo, E. Lodi, and G. M. Pinna, editors, *Theoretical Computer Science*, volume 3701 of *Lecture Notes in Computer Science*, pages 237–250. Springer, 2005.
72. G. Franco, C. Giagulli, C. Laudanna, and V. Manca. DNA Extraction by XPCR. In *[64]*, pages 104–112, 2005.
73. G. Franco and N. Jonoska. Forbidding - Enforcing Conditions in DNA Self-Assembly. In J. Chen, N. Jonoska, and G. Rozenberg, editors, *Nanotechnology, Science and Computation*, pages 105–118, 2005.
74. G. Franco and N. Jonoska. Forbidding-Enforcing Graphs for Self-Assembly. *Congressus Numerantium*, 177:51–63, 2005.
75. G. Franco and V. Manca. A Membrane System for the Leukocyte Selective Recruitment. In C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, Revised Papers of WMC 2003, Tarragona, Spain*, volume 2933 of *Lecture Notes in Computer Science*, pages 181–190. Springer, 2004.
76. G. Franco and V. Manca. An Algorithmic Analysis of DNA Structure. *Soft Computing*, 9(10):761–768, 2005.
77. G. Franco, V. Manca, C. Giagulli, and C. Laudanna. DNA Recombination by XPCR. In *[32]*, pages 55–66, 2006.
78. F. Freund, R. Freund, M. Oswald, M. Margenstern, Y. Rogozhin, and S. Verlan. P Systems with Cutting/Recombination Rules Assigned to Membranes. In C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane*

Computing, volume 2933 of *Lecture Notes in Computer Science*, pages 191–202. Springer, 2004.
79. R. Freund, L. Kari, and Gh. Păun. DNA Computing Based on Splicing: The Existence of Universal Computers. *Theory of Computing Systems*, 32(1):69–112, 1999.
80. R. Freund, Gh. Păun, G. Rozenberg, and A. Salomaa. Watson-Crick Automata. Techn. Report 97-13, Dept of Computer Science, Leiden University, 1997.
81. R. Freund, Gh. Paun, G. Rozenberg, and A. Salomaa, editors. *Membrane Computing, International Workshop, WMC6, Vienna, Austria, 2005, Revised Selected and Invited Papers*, volume 3850 of *Lecture Notes in Computer Science*. Springer, 2006.
82. P. Frisco. *Theory of Molecular Computing - Splicing and Membrane Systems*. PhD thesis, Leiden Institute of Advanced Computer Science (LIACS), Leiden, The Netherlands, June 2004.
83. P. Frisco and R. T. Gibson. A Simulator and an Evolution Program for Conformon-P Systems. In *Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 427–430. IEEE Computer Society Press, 2005.
84. P. Frisco, C. Henkel, and S. Tengely. An Algorithm for SAT without an Extraction Phase. In *[32]*, pages 67–80, 2006.
85. R. H. Garrett and C. M. Grisham. *Biochemistry*. Saunders College Publishing, Harcourt, 1997.
86. M. Garzon, R. Deaton, and D. Reanult. Virtual Test Tubes: A New Methodology for Computing. In *Seventh International Symposium String Processing and Information retrieval, A Coruňa, Spain*, pages 116–121. IEEE Computing Society Press, 2000.
87. M. H. Garzon, K. V. Bobba, and B. P. Hyde. Digital Information Encoding on DNA. In *[117]*, pages 152–166, 2004.
88. M. H. Garzon, N. Jonoska, and S. A. Karl. The Bounded Complexity of DNA Computing. *[125]*, pages 63–72, 1999.
89. A. Gehani and J. Reif. Micro Flow Bio-molecular Computation. *[125]*, pages 197–216, 1999.
90. I. P. Gent and T. Walsh. The Satisfiability Constraint Gap. *Artificial Intelligence*, 81:59–80, 1996.
91. F. J. Ghadessy, J. L. Ong, and P. Holliger. Directed Evolution of Polymerase Function by Compartmentalized Self-Replication. *Proceedings of the National Academy of Science of USA (PNAS)*, 98(8):4552–4557, 2001.
92. P. J. E. Goss and J. Peccoud. Quantitative Modeling of Stochastic Systems in Molecular Biology by Using Stochastic Petri Nets. *Proceedings of the National Academy of Sciences of USA (PNAS)*, 95(12):6750–6755, 1998.
93. M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, A. Riscos-Núñez, and F. J. Romero-Campero. On the Power of Dissolution in P Systems with Active Membranes. In *[81]*, pages 224–240, 2006.
94. M. Hagiya, M. Arita, D Kiga, K. Sakamoto, and S. Yokoyama. Towards Parallel Evaluation and Learning of Boolean μ-Formulas with Molecules. In *[204]*, pages 57–72, 1998.
95. M. Hagiya and A. Ohuchi, editors. *DNA Computing: 8th International Workshop on DNA-Based Computers, DNA8 Sapporo, Japan, June 10-13, 2002. Revised papers*, volume 2568 of *Lecture Notes in Computer Science*. Springer, 2003.
96. J. Hartmanis. On the weight of computation. *Bulletin of the EATCS*, 55:136–138, February 1995.
97. T. Head. Formal Language Theory and DNA: An Analysis of the Generative Capacity of Specific Recombinant Behaviors. *Bulletin of Mathematical Biology*, 49:737–759, 1987.
98. T. Head. Splicing Schemes and DNA. In *Lindenmayer Systems, Impacts on Theoretical Computer Science and Developmental Biology*, pages 371–383. Springer, 1992.

99. T. Head. Splicing languages generated with one sided contexts. In Gh.Păun, editor, *Biomolecular Computing - Theory and Experiments*. Springer, 1998.
100. T. Head. Splicing Representations of Strictly Locally Testable Languages. *Discrete Applied Mathematics*, 87:139–147, 1998.
101. T. Head. Circular Suggestions for DNA Computing. *Pattern Formation in Biology, Vision and Dynamics*, pages 325–335, 2000.
102. T. Head, X. Chen, M. J. Nichols, M. Yamamura, and S. Gal. Aqueous Solutions of Algorithmic Problems: Emphasizing Knights on a 3X3. In *[120]*, pages 191–202, 2002.
103. T. Head, D. Pixton, and E. Goode. Splicing Systems: Regularity and Below. In *[95]*, pages 262–268, 2003.
104. T. Head, G. Rozenberg, R. S. Bladergroen, C. K. D. Breek, P. H. M. Lommerse, and H. P. Spaink. Computing with DNA by Operating on Plasmids. *BioSystems*, 57:87–93, 2000.
105. S. N. Ho, H. D. Hunt, R. M. Horton, J. K. Pullen, and L. R. Pease. Site-Directed Mutagenesis by Overlap Extension Using the Polymerase Chain Reaction. *Gene*, 77(1):51–59, 1989.
106. S. Hussini, L. Kari, and S. Konstantinidis. Coding Properties of DNA Languages. In *[120]*, pages 57–69, 2002.
107. B. Hyrup and P. E. Nielsen. Peptide Nucleic Acids (PNA): Synthesis, Properties and Potential Applications. *Bioorganic and Medical Chemistry*, 4(1):5–23, 1996.
108. Z. Ibrahim. Towards Solving Weighted Graph Problems by Direct-Proportional Length-Based DNA Computing. Research report, Walter J Karplus Summer Research Grant, IEEE Computational Intelligence Society (CIS), September 2004.
109. Z. Ibrahim, Y. Tsuboi, and O. Ono. Direct-Proportional Length-Based DNA Computing for Shortest Path Problem. *International Journal of Computer Science and Applications*, 1(1):46–60, 2004.
110. J. Jonoska and S. A. Karl. A Molecular Computation of the Road Coloring Problem. In *[138]*, pages 87–96, 1996.
111. N. Jonoska. Trends in Computing with DNA. *Journal of Computer Science and Technology*, 19(1):98–113, 2004.
112. N. Jonoska, S. A. Karl, and M. Saito. Three Dimensional DNA Structures in Computing. *[125]*, pages 143–153, 1999.
113. N. Jonoska and K. Mahalingam. Languages of DNA Based Code Words. In J Chen and J. Reif, editors, *DNA Computing, Proceedings of the 9th International Workshop on DNA Based Computers*, volume 2943 of *Lecture Notes in Computer Science*, pages 61–73. Springer, 2004.
114. N. Jonoska and K. Mahalingam. Methods for Constructing Coded DNA Languages. In *[117]*, pages 241–253, 2004.
115. N. Jonoska and M. Margenstern. Tree Operations in P Systems and Lambda-Calculus. *Fundamenta Informaticae*, 59(1):67–90, 2004.
116. N. Jonoska and G.L. McColm. A Computational Model for Self-Assembling Flexible Tiles. In C. S. Calude, M. J. Dinneen, Gh. Păun, M. J. Pérez-Jiménez, and G. Rozenberg, editors, *Unconventional Computation (4th International Conference, Sevilla, Spain, October 2005, proceedings)*, volume 3699 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2005.
117. N. Jonoska, Gh. Păun, and G. Rozenberg, editors. *Aspects of Molecular Computing: Essays Dedicated to Tom Head, on the Occasion of His 70th Birthday*, volume 2950 of *Lecture Notes in Computer Science*. Springer, 2004.
118. N. Jonoska, P. Sa-Ardyen, and N.C. Seeman. Computation by Self-Assembly of DNA Graphs. *Journal of Genetic Programming and Evolvable Machines*, 4(2):123–137, 2003.

119. N. Jonoska and M. Saito. Boundary Components of Thickened Graphs. In *[120]*, pages 70–81, 2002.
120. N. Jonoska and N. C. Seeman, editors. *DNA Computing. 7th International Workshop on DNA-Based Computers, DNA 7. Tampa, FL, USA, June 2001. Revised Papers*, volume 2340 of *Lecture Notes in Computer Science*. Springer, 2002.
121. R. Jung, K. Soondrum, and M. Neumaier. Quantitative PCR. *Clin Chem Lab Med*, 38(9):833–836, 2000. Paper presented at the International IFCC-Roche Conference in Kyoto, Japan, April 2000.
122. P. D. Kaplan, Q. O. Ouyang, D. S. Thaler, and A. Libchaber. Parallel Overlap Assembly for the Construction of Computational DNA Libraries. *Journal of Theoretical Biology*, 188(3):333–341, 1997.
123. L. Kari. DNA Computing: the Arrival of Biological Mathematics. *The Mathematical Intelligencer*, 19(2):9–22, 1997.
124. L. Kari, Gh. Păun, G. Rozenberg, A. Salomaa, and S. Yu. DNA Computing, Sticker Systems, and Universality. *Acta Informatica*, 35(5):401–420, 1998.
125. L. Kari, H. Rubin, and D. H. Rubin, editors. *Bio Systems - Special Issue: Proceedings of The Fourth International Meeting on DNA based Computers*, volume 52, 1999.
126. L. Kari and G. Thierrin. Contextual Insertions/Deletions and Computability. *Information and Computation*, 131(1):47–61, 1996.
127. R. M. Karp. Mapping the Genome: Some Combinatorial Problems Arising in Molecular Biology. In *Proceedings of 25th Annual Symposium on the Theory of Computing, San Diego, CA, USA, May 16-18*, New York, NY, USA: ACM, pages 278–285, 1993.
128. R. M. Karp, C. Kenyon, and O. Waarts. Error-resilient DNA Computation. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, Providence, RI*, AMS/SIAM, pages 458–467, 1996.
129. S. Kashiwamamura, M. Yamamoto, A. Kameda, T. Shiba, and A. Ohuchi. Hierarchical DNA Memory Based on Nested PCR. In *[95]*, pages 112–123, 2004.
130. T. Kazic. Elements of a More Comprehensive Theory of Computing. *[125]*, pages 111–122, 1999.
131. J. Khodor and D. K. Gifford. Design and Implementation of Computational Systems Based on Programmed Mutagenesis. *[125]*, pages 93–97, 1999.
132. S. Kirkpatrick and B. Selman. Critical Behaviour in the Satisfiability of Random Boolean Expressions. *Science*, 264(5163):1297–1301, 1994.
133. S. C. Kleene. Representation of Events in Nerve Nets and Finite Automata. In *Automata Studies*, pages 3–42. Princeton University Press, Princeton, NJ, 1956.
134. K. Komiya, K. Sakamoto, H. Gouzu, S. Yokoyama, M. Arita, A. Nishikawa, and M. Hagiya. Successive State Transitions with I/O Interface by Molecules. In *[44]*, pages 17–26, 2001.
135. J. Kuramochi and Y. Sakakibara. Intensive *in vitro* Experiments of Implementing and Executing Finite Automata in Test Tube. In *[32]*, pages 193–202, 2006.
136. T. H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J. H. Reif, and N. C. Seeman. Construction, Analysis, Ligation, and Self-Assembly of DNA Triple Crossover Complexes. *Journal of the American Chemical Society*, 122(9):1848–1860, 2000.
137. L. Landweber. RNA Based Computing: Some Examples from RNA Catalysis and RNA Editing. In *[138]*, pages 181–189, 1996.
138. L. F. Landweber and E. B. Baum, editors. *DNA Based Computers II*, volume 44 of *DIMACS Series AMS*, 1996.
139. L. F. Landweber and L. Kari. The Evolution of Cellular Computing: Nature's Solution to a Computational Problem. *[125]*, pages 3–13, 1999.
140. J. Y. Lee, H-W. Lim, S-I. Yoo, B-T. Zhang, and T. H. Park. Efficient Initial Pool Generation for Weighted Graph Problems Using Parallel Overlap Assembly. In *[64]*, pages 215–223, 2005.

141. D. Li. Is DNA Computing Viable for 3-SAT Problems? *Theoretical Computer Science*, 290(3):2095–2107, 2003.
142. A. Lindenmayer. Mathematical Models for Cellular Interaction in Development. Part I and Part II. *Journal of Theoretical Biology*, 18(3):280–315, 1968.
143. R. J. Lipton. DNA Solutions of Hard Computational Problems. *Science*, 268(5210):542–545, 1995.
144. R. J. Lipton and E. B. Baum, editors. *DNA Based Computers*, volume 27 of *DIMACS Series AMS*, 1995.
145. S. Liu, D. S. Thaler, and A. Libchaber. Signal and Noise in Bridging PCR. *BMC Biotechnology*, 2(13), 2002.
146. V. Manca. On Some Forms of Splicing. In *Where Mathematics, Computer Science, Linguistics and Biology meet*, pages 387–398. Kluwer Academic Publishers, 2001.
147. V. Manca. A Proof of Regularity for Finite Splicing. In *[117]*, pages 309–318, 2004.
148. V. Manca. On the Logic and Geometry of Bilinear Forms. *Fundamenta Informaticae*, 64:257–269, 2005.
149. V. Manca, L. Bianco, and F. Fontana. Evolution and Oscillation of P systems: Applications to Biological Phenomena. In G. Mauri, Gh. Păun, M. J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 5th International Workshop, WMC 2004*, volume 3365 of *Lecture Notes in Computer Science*, pages 63–84. Springer, 2005.
150. V. Manca and G. Franco. Computing by Polymerase Chain Reaction. Submitted.
151. V. Manca, G. Franco, and G. Scollo. State Transition Dynamics – Basic Concepts and Molecular Computing Perspectives. In M. Gheorghe, editor, *Molecular Computational Models - Unconventional Approaches*, chapter 2, pages 32–55. Idea Group, 2005.
152. V. Manca, C. Martin-Vide, and Gh. Păun. New Computing Paradigms Suggested by DNA Computing: Computing by Carving. *[125]*, pages 47–54, 1999.
153. V. Manca and M. D. Martino. From String Rewriting to Logical Metabolic Systems. In Gh. Păun and A. Salomaa, editors, *Grammatical Models of Multi-agent Systems*, pages 297–315. Gordon and Breach Science Publishers, London, 1999.
154. V. Manca and C. Zandron. A Clause String DNA Algorithm for SAT. In *[120]*, pages 172–181, 2002.
155. C. Mao. The Emergence of Complexity: Lessons from DNA. *PLoS Biology*, 2(12):e431, 2004.
156. C. Mao, T. LaBean, J.H. Reif, and N.C. Seeman. Logical Computation Using Algorithmic Self-Assembly of DNA Triple Crossover Molecules. *Nature*, 407:493–496, 2000.
157. C. Mao, W. Sun, and N.C. Seeman. Assembly of Borromean Rings from DNA. *Nature*, 386:137–138, 1997.
158. C. Mao, W. Sun, and N.C. Seeman. Designed Two-dimensional Holliday Junction Arrays Visualized by Atomic Force Microscopy. *Journal of the American Chemical Society*, 121:5437–5443, 1999.
159. M. Margenstern. *Une Machine de Turing Universelle Non-effaçante à Exactement Trois Instructions Gauches*, pages 101–106. Série I. C. R. Acad. Sci. Paris t. 320, 1995.
160. M. Margenstern. On Quasi-Unilateral Universal Turing Machines. *Theoretical Computer Science*, 257(1-2):153–166, 2001.
161. M. Margenstern and L. Pavlotskaïa. *Deux Machines de Turing Universelles à au Plus Deux Instructions Gauches*, pages 1395–1400. Série I. C. R. Acad. Sci. Paris, t.320, 1995.
162. M. Margenstern and L. Pavlotskaïa. On the Optimal Number of Instructions for Universal Turing Machines Connected with a Finite Automaton. In M. Kutrib and

T. Worsch, editors, *Pre-proceedings of Cellular Automata Workshop, March 25-27 1996, Schloss Rauischholzhausen, Germany*, 1996.
163. J. L. Marx. Multiplying Genes by Leaps and Bounds. *Science*, 240(4858):1408–1410, 1988.
164. A. Mateescu, Gh. Păun, G. Rozenberg, and A. Salomaa. Simple Splicing Systems. *Discrete Applied Mathematics*, 84(1-3):145–163, 1998.
165. H. Matsuno, R. Murakani, R. Yamane, N. Yamasaki, S. Fujita, J. Yoshimori, and S. Miyano. Boundary Formation by Notch Signaling in Drosophila Multicellular Systems: Experimental Observations and Gene Network Modeling by Genomic Object Net. In R. Bl. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing, Singapore*, pages 152–163. World Scientific Press, 2003.
166. W. S. McCulloch and W. H. Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.
167. R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, Cambridge, Massachusetts, 1971.
168. R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining Computational Complexity from Characteristic 'Phase Transitions'. *Nature*, 400:133–137, 1999.
169. N. Morimoto, M. Arita, and A. Suyama. Solid Phase DNA Solution to the Hamiltonian Path Problem. In *[204]*, pages 193–206, 1998.
170. K. Mullis and F. Faloona. Specific Synthesis of DNA *in vitro* via a Polymerase-Catalyzed Chain Reaction. *Methods Enzymol*, 155:335–350, 1987.
171. K. Mullis, F. Faloona, S. Scharf, R. Saiki, G. Horn, and H. Erlich. Specific Enzymatic Amplification of DNA *in vitro*: the Polymerase Chain Reaction. *Cold Spring Harbor Symposium Quantitative Biology*, 51:263–273, 1986.
172. M. Nagasaki, S. Onami, S. Miyano, and H. Kitano. Bio-calculus: Its Concept and Molecular Interaction. In *Genome Informatics Workshop, December 14-15, Tokyo, Japan*, volume 10, pages 133–143, 1999.
173. H. Nakagawa, K. Sakamoto, and Y. Sakakibara. Development of an *in vivo* Computer Based on *Escherichia Coli*. In *[32]*, pages 203–212, 2006.
174. K. Onodera and T. Yokomori. Linearizer and Doubler: Two mappings to Unify Molecular Computing Models Based on DNA Complementarity. In *[32]*, pages 224–235, 2006.
175. Q. Ouyang, P. D. Kaplan, S. Liu, and A. Libchaber. DNA Solution of the Maximal Clique Problem. *Science*, 278:446–449, 1997.
176. A. Păun and Gh. Păun. The Power of Communication: P Systems with Symport/Antiport. *New Generation Computing*, 20(3):295–306, 2002.
177. Gh. Păun. On Semicontextual Grammars. *Bull. Math. Soc. Sci. Math. Roumanie*, 28(76):63–68, 1984.
178. Gh. Păun. Computing with Membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000. Turku Center for Computer Science-TUCS Report 208, November 1998, www.tucs.fi.
179. Gh. Păun. DNA Computing Based on Splicing: Universality Results. *Theoretical Computer Science*, 231(2):275–296, 2000.
180. Gh. Păun. *Membrane Computing. An Introduction*. Springer, 2002.
181. Gh. Păun. From Cells to Computers: Membrane Computing - A Quick Overview. In *[64]*, pages 268–289, 2005.
182. Gh. Păun and G. Rozenberg. Sticker Systems. *Theoretical Computer Science*, 204(1):183–203, 1998.
183. Gh. Păun and G. Rozenberg. A Guide to Membrane Computing. *Theoretical Computer Science*, 287:73–100, 2002.

184. Gh. Păun, G. Rozenberg, and A. Salomaa. Computing by Splicing: Programmed and Evolving Splicing Systems. In *IEEE International Conference on Evolutionary Computation. Special Session on DNA Based Computation*, pages 273–277. Indiana University, Purdue University, Indianapolis, Illinois, USA, 1997.
185. Gh. Păun, G. Rozenberg, and A. Salomaa. *DNA Computing. New Computing Paradigms*. Springer, 1998.
186. Gh. Păun, G. Rozenberg, and T. Yokomori. Hairpin Languages. *International Journal of Foundations of Computer Science*, 12(6):837–847, 2001.
187. Gh. Păun, Y. Suzuki, and H. Tanaka. P Systems with Energy Accounting. *International Journal of Computer Mathematics*, 78(3):343–364, 2001.
188. M. J. Pérez-Jiménez and F. Sancho-Caparrini. Solving Knapsack Problems in a Sticker Based Model. In *[120]*, pages 161–171, 2002.
189. V. Phan and M. H. Garzon. The Capacity of DNA for Information Encoding. In *[64]*, pages 281–292, 2005.
190. D. M. Prescott, A. Ehrenfeucht, and G. Rozenberg. Molecular Operations for DNA Processing in Hypotrichous Ciliates. *European Journal of Protistology*, 37:241–260, 2001.
191. D. M. Prescott, A. Ehrenfeucht, and G. Rozenberg. Template-guided Recombination for IES Elimination and Unscrambling of Genes in Stichotrichous Ciliates. *Journal of Theoretical Biology*, 222(3):323–330, 2003.
192. A. Regev, E. M. Panina, W. Silverman, L. Cardelli, and E. Shapiro. BioAmbients: An Abstraction for Biological Compartments. *Theoretical Computer Science – Special Issue on Computation Methods in Systems Biology*, 325(1):141–167, 2004.
193. J. H. Reif, T. H. LaBean, M. Pirrung, V. S. Rana, B. Guo, C. Kingsford, and G. S. Wickham. Experimental Construction of Very Large Scale DNA Databases with Associative Search Capability. In *[120]*, pages 231–247, 2002.
194. J. H. Reif, S. Sahu, and P. Yin. Complexity of Graph Self-Assembly in Accretive Systems and Self-Destructive Systems. In *[32]*, pages 257–274, 2006.
195. Y. Rogozhin. Small Universal Turing Machines. *Theoretical Computer Science*, 168:215–240, 1996.
196. J. A. Rose, R. J. Deaton, M. Hagiya, and A. Suyama. PNA-Mediated Whiplash PCR. In N. Jonoska and N. C. Seeman, editors, *[120]*, pages 104–116, 2002.
197. J. A. Rose, M. Hagiya, R. J. Deaton, and A. Suyama. A DNA-based in Vitro Genetic Program. *Journal of Biological Physics*, 28(3):493–498, 2002.
198. P. W. K. Rothemund. A DNA and Restriction Enzyme Implementation of Turing Machines. In *[144]*, pages 75–120, 1995.
199. P. W. K. Rothemund, N. Papadakis, and E. Winfree. Algorithmic Self-Assembly of DNA Sierpinski Triangles. *PLoS Biology*, 2(12):2041–2053, 2004.
200. S. Roweis and E. Winfree. On the Reduction of Errors in DNA Computation. *Journal of Computational Biology*, 6(1):65–75, 1999.
201. G. Rozenberg. Computer Science, Informatics, and Natural Computing. *Bulletin of the EATCS*, 85:133–134, February 2005.
202. G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*. 3 volumes. Springer, 1997.
203. G. Rozenberg and H. Spaink. DNA Computing by Blocking. *Theoretical Computer Science*, 292:653–665, 2003.
204. H. Rubin and D. H. Wood, editors. *DNA Based Computers III. University of Pennsylvania, June 23-25*, volume 48 of *DIMACS Series AMS*, 1997.
205. W. Rychlik, W.J. Spencer, and R.E. Rhoads. Optimization of the Annealing Temperature for DNA Amplification *in vitro*. *Nucleic Acids Research*, 18(21):6409–6412, 1990.

206. P. Sa-Ardyen, N. Jonoska, and N. C. Seeman. Self-Assembly of Graphs Represented by DNA Helix Axis Topology. *Journal of American Chemical Society*, 126(21):6648–6657, 2004.
207. R. K. Saiki, S. Scharf, F. Faloona, K. B. Mullis, G. T. Horn, H. A. Erlich, and N. Arnheim. Enzymatic Amplification of β-Globin Genomic Sequences and Restriction Site Analysis for Diagnosis of Sickle Cell Anemia. *Science*, 230:1350–1354, 1985.
208. Y. Sakakibara and C. Ferretti. Splicing on Tree-like Structures. In *[204]*, pages 348–358, 1997.
209. K. Sakamoto, H. Gouzu, K. Komiya, D. Kiga, S. Yokoyama, T. Yokomori, and M. Hagiya. Molecular Computation by DNA Hairpin Formation. *Science*, 288(5469):1223–1226, 2000.
210. K. Sakamoto, D. Kiga, K. Komiya, H. Gouzu, S. Yokoyama, S. Ikeda, H. Sugiyama, and M. Hagiya. State Transitions by Molecules. *[125]*, pages 81–91, 1999.
211. S. J. Scharf, G. T. Horn, and H. A. Erlich. Direct Cloning and Sequence Analysis of Enzymatically Amplified Genomic Sequences. *Science*, 233(4768):1076–1078, 1986.
212. C. Schildkraut and S. Lifson. Dependence of the Melting Temperature of DNA on Salt Concentration. *Biopolymers*, 3(2):195–208, 1965.
213. M. P. Schutzenberger. Sur Certaines Operations de Fermeture dans les Languages Rationels. *Symposium Mathematica*, 15:245–253, 1975.
214. N. C. Seeman. De Novo Design of Sequences for Nucleic Acid Structural Engineering. *Journal of Biomolecular Structure and Dynamics*, 8(3):573–581, 1990.
215. N. C. Seeman, J. H. Chen, and N. R. Kallenbach. Gel Electrophoretic Analysis of DNA Branched Junctions. *Electrophoresis*, 10:345–354, 1989.
216. L.A. Segel and I.R. Cohen. *Design Principles for the Immune System and Other Distributed Autonomous Systems*. Santa Fe Institute Studies in the Sciences of Complexity, Oxford University Press, 2001.
217. B. Selman, D. G. Mitchell, and H. J. Levesque. Generating Hard Satisfiability Problems. *Artificial Intelligence*, 81:17–29, 1996.
218. R. F. Service. How Far Can We Push Chemical Self-Assembly. *Science – Special Issue: What don't we know?*, 309(5731):95, 2005.
219. J. Shendure, G. J. Porreca, N. B. Reppas, X. Lin, J. P. McCutcheon, A. M. Rosenbaum, M. D. Wang, K. Zhang, R. D. Mitra, and G. M. Church. Accurate Multiplex Polony Sequencing of an Evolved Bacterial Genome. *Science*, 309(5741):1728–1732, 2005.
220. R. R. Sinden. Molecular Biology: DNA Twists and Flips. *Nature*, 437:1097–1098, 2005.
221. W. D. Smith. Church's Thesis Meets the N-Body Problem. Technical Report, NEC Research Institute, March 1999.
222. D. Soloveichik and E. Winfree. Complexity of Self-Assembled Shapes. In *[64]*, pages 344–354, 2005.
223. W.P.C. Stemmer. DNA Shuffling by Random Fragmentation and Reassembly: *In vitro* Recombination for Molecular Evolution. *Proceedings of the National Academy of Science of USA (PNAS)*, 91:10747–10751, 1994.
224. Y. Suzuki, Y. Fujiwara, H. Tanaka, and J. Takabayashi. Artificial Life Applications of a Class of P Systems: Abstract Rewriting Systems on Multisets. In C.S. Calude, Gh. Păun, G. Rozenberg, and A. Salomaa, editors, *Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View*, volume 2235 of *Lecture Notes in Computer Science*, pages 299–346. Springer, 2001.
225. N. Takahashi, A. Kameda, M. Yamamoto, and A. Ohuchi. Aqueous Computing with DNA Hairpin-Based RAM. In *[64]*, pages 355–364, 2005.

226. D. S. Tawfik and A. D. Griffiths. Man-made Cell-like Compartments for Molecular Evolution. *Nature Biotechnology*, 16:652–656, 1998.
227. A. M. Turing. On Computable Real Numbers, with an Application to the Entscheidungsproblem. *Proc. London Math. Soc.*, 2(42):230–265, 1936.
228. D. van Noort, F. Gast, and J. S. McCaskill. DNA Computing in Microreactors. In *[120]*, pages 33–45, 2002.
229. D. van Noort, Z.-L. Tang, and L. F. Landweber. Fully Controllable Microfluidics for Molecular Computers. *Journal of the Association for Laboratory Automation (JALA)*, 9(5), 2004.
230. Y. Wang, J.E. Mueller, B. Kemper, and N.C. Seeman. The Assembly and Characterization of 5-arm and 6-arm DNA Junctions. *Biochemistry*, 30:5667–5674, 1991.
231. J. D. Watson, M. Gilman, J. A. Witowski, and M. Zoller. *Recombinant DNA*. W. H. Freeman and Company, New York and Oxford, second edition, 1992.
232. R. Weiss, S. Basu, S. Hooshangi, A. Kalmbach, D. Karig, R. Mehreja, and I. Netravali. Genetic Circuit Building Blocks for Cellular Computation, Communications, and Signal Processing. *Natural Computing*, 2(1):43–84, 2003.
233. N. Wiener. *Cybernetics: or Control and Communication in the Animal and the Machine*. MIT Press, Massachusetts, second edition, 1965.
234. E. Winfee, F. Liu, L. Wenzler, and N. C. Seeman. Design of Self-Assembly of Two-dimensional Crystals. *Nature*, 494:539–544, 1998.
235. E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, Pasadena, California, May 1998.
236. E. Winfree. Whiplash PCR for O(1) Computing. Technical Report, California Institute of Technology, May 1998.
237. E. Winfree, T. Eng, and G. Rozenberg. String Tile Models for DNA Computing by Self-Assembly. In *[44]*, pages 63–88, 2001.
238. E. Winfree and D. K. Gifford, editors. *DNA Based Computers V, Fifth Annual Meeting on DNA Based Computers held at MIT*, volume 54 of *DIMACS Series AMS*, 2000.
239. E. Winfree, X. Yang, and N. Seeman. Universal Computation via Self-Assembly of DNA: Some Theory and Experiments. In *[138]*, pages 191–213, 1996.
240. D. H. Wood, H. Bi, S. O. Kimbrough, D.-J. Wu, and J. Chen. DNA Starts to Learn Poker. In *[120]*, pages 92–103, 2002.
241. M. Yamamura, Y. Hiroto, and T. Matoba. Another Realization of Aqueous Computing with Peptide Nucleic Acid. In *[120]*, pages 213–222, 2002.
242. P. Yin, S. Sahu, A. J. Turberfield, and J. H. Reif. Design of Autonomous DNA Cellular Automata. In *[32]*, pages 399–416, 2006.
243. T. Yokomori. Molecular Computing Paradigm - Toward Freedom from Turing's Charm. *Natural Computing*, 1:333–390, 2002.
244. T. Yokomori, Y. Sakakibara, and S. Kobayashi. A Magic Pot: Self-Assembly Computation Revisited. In W. Brauer, H. Ehrig, J. Karhumki, and A. Salomaa, editors, *Formal and Natural Computing. Essays dedicated to Grzegorz Rozenberg*, volume 2300 of *Lecture Notes in Computer Science*, pages 418–429. Springer, 2002.
245. H. Yoshida and A. Suyama. Solution to 3-SAT by Breath First Search. In *[238]*, pages 9–22, 2000.
246. Y. Zhang and N.C. Seeman. The Construction of a DNA Truncated Octahedron. *Journal of the American Chemical Society*, 116:1661–1669, 1994.

9

Acknowledgments

Everything that is written merely to please the author is worthless.

Blaise Pascal[1]

I did my best to make this dissertation as interesting and pleasant to read as possible. As I was writing, I was spurred on by the challenge to write something new and satisfactory for my advisor, the hope that the reading of the thesis would not be boring for my eminent referees (and thus for any reader) and the ambition that this work might become a reference for undergraduate students interested in molecular computing. However, I could not even have dreamt of achieving any of these goals without a considerable amount of research work, which was only made possible by extensive collaboration with others, and by the support of some people who bolstered my strength during these past four years.

First of all, this work was conceived and realized thanks to the encouragement, the teaching, the inspiration, the notable scientific contribution, and the funds from my advisor Vincenzo Manca, who is able to bring out the best of my capabilities and to appropriately channel my different interests. In the eyes of his collaborators, Vincenzo represents a constant example of deep interest for research, combined with solid values and a respect for human relationships. In my case, he has also been a good and generous teacher with an exceptional breadth of knowledge, an intuitive and passionate researcher, a continuous source of inspiration and enthusiasm for my whole research, a wise friend and a strong leader at the same time. We even fought on some occasions, as having different points of view, but most of the time his strictness turned out to be good to discipline my temperament and my maturing as a researcher. I would like to thank Vincenzo very much for all the discussions we had, and for his way of supervising the research leading to the writing of this dissertation; his 'well-done' was the first important gratification of my work. Above all, I will not forget that he has always believed in me, even before I myself did, and sometimes in spite of my own skepticism.

[1] Quoted in W H Auden and L Kronenberger, The Viking Book of Aphorisms, New York, 1966.

By looking through the introductory chapter of this thesis, or simply by taking a look at the bibliography, one can see what an honour it was for me to have Gheorghe Păun and Grzegorz Rozenberg as referees of my work. It was also a pleasure, and I am grateful to both of them for reading the whole manuscript carefully (see for example footnote 4 of Chapter 2) and writing helpful, generous, and encouraging comments; their detailed and accurate reports have carried huge gratification for my efforts and will represent a significant step of my scientific journey. I do not know Prof. Rozenberg personally, but I thank him very much for the flattering accolade he wrote about my work, it will make me happy for a long time. On the contrary, I met George Păun several times, and, on this occasion, I like to express my gratitude also for his welcome to the membrane computing community (the first time in Curtea de Argeş), for his scientific support (the article [67], for example, arose from a personal communication with him), for his 'promoting behaviour' towards young researchers, and for his passionate explanation of the event during the Romanian Revolution of 1989, when regime soldiers refused to shoot at the universities.

The model for DNA self-assembly proposed in Chapter 7 is "an elegant piece of work"[2] developed with Nataša Jonoska during my visit at the University of South Florida. That was a very important experience for my research and my life, and during that stay I wrote the greater part of this dissertation. I am grateful to Natasha for giving me this opportunity by kindly agreeing to be my host. Working with her was really stimulating, we collaborated on two articles [74, 73], and we started another collaboration with Anna Plaas and Barbara Osborn from the Medical School of USF on the subject described in Section 8.2. I would like to thank them as well, for the time together, and to kindly show us their experimental results (see Figure 8.5). Natasha wrote some pieces of text which I put together in Chapter 7, and she suggested the nice style I adopted for the definitions, in which the concepts are underlined. I learned a lot of things in her office, about scientific literature and research directions, art, English language, American culture. I appreciated as well the Japanese and Indian restaurants where she used to take me! I am also grateful to Gregory McColm for officially inviting me to give a talk at the Department of Mathematics of USF, where the practical arrangements concerning my stay were all taken care of by means of the efficient work of Natasha and her collaborators, Ayako, Beverly, Mary Ann and Nancy.

I am also indebted to a lot of people in the United States who helped me overcome any difficulty I might encounter in everyday life with their warm and helpful presence. Ana Staninska was to me the best person one can hope to find when going to a foreign country, extremely friendly and helpful all the time. I am left with very good memories from the conferences held in Florida Atlantic University (The Thirty-Sixth Southeastern International Conference on Combinatorics, Graph Theory, and Computing) and in University of Western Ontario (The 11th International Meeting on DNA Computing), where I went with Natasha and Ana. Besides, I would like to thank Mintie and Sasha, for showing me how so different cultures (Indian, American and Macedonian) can successfully mix in a couple, Angela and Envar, for their very first and really kind welcome and for being an example of strength and simplicity, Ferenc, for his being so solid and good, Djibi,

[2] As written by one of my referees.

for explaining to me a few details of Muslim culture, Matt and Hannah for bringing the culture from Texas and Colorado to my eyes, and Ron and Jeanne as well as Elka and Mende, for concretely showing me that chronological age has nothing to do with having fresh and youthful thoughts. Finally, I like to thank my affectionate cousins in the States (Emilia, Roberto, Sandra, Adriano..) for their kind support, and all my flatmates for their patience during the writing of this thesis, Anna and Lucia in Verona, Sara, Sarah and Mohini in Tampa.

My level of spoken English required a lot of patience from all these people, whereas the level of my written English in this dissertation was definitively raised by the remarkable work of Matthias Rath, Giuseppe Scollo, and Antoine Vella, who corrected and polished the text passages of all the introductory parts. They did a kind of magic transformation, by putting all my ideas into a fluent and good English text, while completely maintaining my own way to present them. I wish I had the appropriate words to thank them, both for their actual work on this thesis and for all the other support they promptly gave me.

During the last years of PhD, Matthias made me have a regular life rhythm by taking me into his world of beautiful music, of pure interest for art and reading, of enchanting pictures, of delicious dinners. I am really grateful, for all the times when he kept me up with his strength and his brilliant irony, for his unfaltering and convincing way of believing in me, and for what I learned, even about myself, by his impressive personality. Giuseppe (shortly 'Pippo') was extremely nice during the entire course of my PhD studies, and very helpful for my work right at the deadlines related to the dissertation. With Vincenzo Manca and Roberto Segala he was a member of the committee following the development of my thesis, and I thank all of them. Roberto was good to suggest the selection of just some topics, among my research interests, for the dissertation; Pippo and Vincenzo suggested even the title; they all were very careful and encouraging throughout the three years of work. I had the opportunity to collaborate with Pippo on concepts of discrete dynamics such as recurrence and attractors [151], and it was an educative experience to me; he showed me that it is really possible to stay focused and patient even after many many hours of work! Antoine is my favourite English teacher, but he is also one of the kindest people I ever met. I would like to thank him for opening my mind, for showing me how spontaneous research can be (we were studying together "superior analysis"..), for his help to make good decisions, such as that regarding my PhD, for all the times he supported me by his extraordinary sensibility, and for his being such a gentleman in every occasion.

The experiments reported in Chapter 6 were carried out at the laboratories of the Department of Pathology at the University of Verona, in the course of a collaborative work with Cinzia Giagulli and Carlo Laudanna. I thank Cinzia for the pleasant working together and for her patiently explaining me some tricks, coming from experience, that are hidden behind the practical execution of experiments; I thank Carlo for supervising our experiments, and for explaining us in layman's terms the leukocyte selective recruitment process, which prompted the work [75].

Federico Fontana and Luca Bianco in the last years enriched the molecular computing group in Verona. We fruitfully collaborated on maximum search algorithms for membrane systems [67] and on modelling biological processes with membrane systems [21]. I thank them for the experiences together and for being

a constant example of hard and determined work; in particular I am grateful to Luca for writing the program that allowed us to check the primers involved in the first experiments (in the way described in Section 3.5). I also wish to thank some anonymous referees for their extensive comments and suggestions for improvements on my work (see for example Section 6.4.1), and all those people with whom I had stimulating discussions: Daniela Besozzi (on biological aspects of membranes), Eshel Ben-Jacob (on bacteria organization), Matteo Cavaliere (on different ways to see computation), Pierluigi Frisco (on P-conformons), Daniela Genova (on enforcing-forbidding systems), Marian Gheorghe (on applications of membrane computing to cell populations), Maurice Margenstern (on DNA duplication and computationally universal systems), Fernando Sancho-Caparrini (on self-assembly and membranes), Ned Seeman (on art and self-assembly) and Claudio Zandron (on self-assembly and parallel computations). A special 'thank you' goes to Gyorgy Vaszil, who was very helpful during my whole stay in Hungary, on the occasion of the conferences DLT7 and DCFS.

My affiliation is the Department of Computer Science of the University of Verona, where I could comfortably develop my research in good working conditions, owing to the efforts of the departmental staff. For this support, I would like to thank our dear director Vittorio Murino and our sweet and efficient secretaries, Anna Bruttomesso and Aurora Miorelli. I'm grateful as well to both Roberto Giacobazzi and Paolo Fiorini for their friendly first welcome, to Andrea Masini for his suggestions and the financial support to send this dissertation to Spain, to the Netherlands and to Colorado, to Maria Paola Bonacina for our nice conversations after work, to Fausto Spoto for giving me the first technical explanations of basic programming, and to Massimo Merro for his nice way of supporting me with wisdom and sincere friendship.

My doctorate progressed smoothly also thanks to the first advice of Debora Botturi and Isabella Mastroeni (with whom I shared the bizarre experience of attending departmental meetings..), and to the teamwork with my colleagues (Damiano, Marco, Rosalba, ...) all the times we shared exams or deadlines. Among my PhD colleagues, particular places are occupied by Roberto Bonato, since we nicely spent many hours in the same office while writing our dissertation, and Amalia Degötzen, who is a good friend of mine. She is a rare combination of strength and tenderness; we shared more than work stuff and I feel lucky for this. Another good friend with whom I happily share the office is Francesca Mantese, who is always full of energy and good humour. I am grateful to both of them for making me feel at home all the times we are together.

Finally, my gratitude goes doubtless to my old friends and to my family. Chiara, Claudio and Pietro are always in my life, and it is important to know that they are still there for any kind of support I might need. My auntie Antonella greatly encouraged me to start a PhD, she is a very sweet and special person, enthusiastic of all my 'academic successes'. My parents and my brother are always in my heart, wherever I go, they represent a point of reference and a concrete example of honesty and strength to which I often turn. I thank very much especially my mum for her generosity, without which I likely could never have been a PhD student, and writing this dissertation would not have been possible.

10

Sommario

Finito questo, l'alta corte santa
risonò per le spere un "Dio laudamo"
ne la melode che là sù si canta.

Dante Alighieri [1]

Questa tesi raccoglie i risultati della mia ricerca nell'ambito del calcolo molecolare, e li presenta in un ordine che rispetta quello cronologico in cui sono stati conseguiti. La maggior parte dei contenuti originali, che vengono proposti nei sei capitoli centrali, sono già stati pubblicati o sono in attesa di pubblicazione, in articoli di cui vengono riportati gli estremi alla fine dei capitoli corrispondenti. Segue un sommario delle idee che sono state sviluppate nella tesi, generalmente frutto di collaborazioni con il relatore ed il suo gruppo, ma anche con gruppi afferenti ad altri dipartimenti (di informatica, matematica, o medicina).

Il primo capitolo è di introduzione all'area di ricerca e allo stato dell'arte, sia del calcolo molecolare che in particolare del calcolo DNA. I problemi affrontati nella tesi sono inseriti nel quadro generale del calcolo col DNA, di cui vengono evidenziate le tendenze più recenti. Il paragrafo conclusivo dell'introduzione suggerisce sia qualche riferimento bibliografico a chi volesse approfondire le conoscenze nel campo, che alcune risposte a curiosità che sono emerse da domande di amici o colleghi nel corso dei miei studi (per esempio, "come si definisce la complessità di un algoritmo biologico?", "quanto costa un esperimento di DNA computing?", "da dove viene il DNA utilizzato negli esperimenti?").

Nel secondo capitolo vengono individuate delle proprietà algoritmiche e computazionali delle molecole DNA, che indicano una nuova prospettiva di speculazione, con ispirazioni di ricerca sia in biologia che in informatica. Si parte dall'analizzare la struttura e l'interazione di stringhe fluttuanti in un supporto fluido, quali sono quelle biologiche. La bilinearità (del doppio filamento che costituisce la molecola di DNA) la complementarietà (di Watson-Crick tra le basi appaiate lungo i due filamenti) e l'antiparallelismo (dato dalla direzione opposta

[1] vv. 112-114, Paradiso XXIV, Divina Commedia, 1320.

dei due filamenti DNA) risultano requisiti necessari per la realizzazione di un algoritmo efficiente di duplicazione di stringhe fluttuanti, quale quello implementato nel nucleo delle cellule per duplicare il corredo cromosomico. Tale algoritmo di duplicazione viene formalizzato, confrontato in complessità con altri tre, ed esteso ad un sistema computazionale basato su regole di riscrittura definite su doppie stringhe e aventi una possibile implementazione biotecnologica. L'universalità di questo sistema dimostra ancora una volta come tutto il calcolabile possa essere ottenuto dalla combinazione di poche operazioni (enzimatiche) su doppie stringhe di DNA. A concludere il capitolo troviamo una presentazione generale dei sistemi a membrana, un modello di calcolo molecolare proposto in [178], di cui viene richiamata una strategia di applicazione delle regole di evoluzione, formulata recentemente [23, 21, 149], che sostituisce il concetto di trasformazione di oggetti individuali con quello di evoluzione dinamicamente controllata di popolazioni di oggetti. In questo contesto, viene brevemente descritto un sistema a membrana che modella il processo immunologico del reclutamento selettivo di leucociti.

I quattro capitoli centrali costituiscono il cuore della tesi. Nelle nostre ricerche sulle computazioni DNA, dove i risultati teorici hanno trovato riscontro negli esperimenti di laboratorio, abbiamo realizzato una variante della PCR ('Polymerase Chain Reaction': una delle più famose e usate tecniche di biologia molecolare), chiamata XPCR, che implementa regole splicing di tipo "null context" [97], e che costituisce una base implementativa per nuove procedure di manipolazione di DNA: sia di generazione, che di estrazione, che di mutagenesi. La tecnica XPCR è stata testata in diverse situazioni, e gli esperimenti di laboratorio che l'hanno validata sono tutti riportati nel Capitolo 6. La tecnologia semplice che è stata introdotta con questo approccio è risultata interessante di per sé, e ha trovato diverse applicazioni in contesti biologici, che vanno al di là dei problemi di DNA computing che ne hanno motivato la ricerca. La XPCR ha i vantaggi e l'efficienza di una tecnica di estensione enzimatica, e nello stesso tempo si è dimostrata conveniente rispetto ad altri metodi in letteratura in termini di velocità e realizzabilità.

I bio-algoritmi basati su XPCR sia hanno interessanti proprietà combinatoriche, e sia hanno dato origine a procedure biotecnologiche facili da implementare. Nel Capitolo 3, dopo un'analisi del processo PCR, è stato proposto un nuovo metodo di estrazione DNA che si avvale principalmente dell'azione della polimerasi. La sua complessità computazionale è stata successivamente ridimensionata, in [71], e il suo rendimento è stato migliorato, almeno a livello teorico, in [150], dove se ne trova una variante che evita le chimere, assieme all'algoritmo di mutagenesi che è descritto nel Capitolo 5. Nel Capitolo 4 abbiamo sviluppato un nuovo metodo per implementare in vitro il 'cross-over' parallelo rispetto ad una sottosequenza, con la specifica di generare librerie combinatoriche di stringhe, utili sia per il calcolo DNA che per studi di interesse biologico. Il confronto con la letteratura si può ridurre a quello presentato nella Sezione 4.5 rispetto al metodo "parallel overlap assembly (POA)" [223], che implementa un cross-over tra sequenze omologhe guidato da ripetuti cicli di ibridizzazione ed estensione tramite polimerasi. Nel Capitolo 5 viene presentato un bio-algoritmo (non ancora convalidato sperimentalmente) per risolvere un'istanza del famoso problema SAT con dimensioni significative. L'approccio è essenzialmente quello classico della forza bruta, ma questo algoritmo potenzialmente migliora i metodi esistenti, sia nella

fase di generazione, che richiede solo due sequenze iniziali ed ha una complessità logaritmica, che in quella di estrazione, implementata da una combinazione di operazioni di taglio (azione di un'endonucleasi di restrizione) ed estensione di stringhe (azione di polimerasi).

Infine, abbiamo esteso il nostro interesse dall'aggregazione (nel seguito "self-assembly") lineare di stringhe DNA al processo di self-assembly tridimensionale di molecole DNA. Con l'espressione *self-assembly* s'intende un processo in cui delle sottostrutture si aggregano spontaneamente in sovrastrutture, secondo un'affinità propria delle sottostrutture. Nel caso del DNA l'affinità è quella dettata dall'appaiamento Watson-Crick di sequenze, e le strutture che si formano devono soddisfare vincoli fisici e chimici, formalizzati nel Capitolo 7 da un insieme di regole di tipo "forbidding and enforcing" [55]. In particolare, una famiglia di grafi viene proposta come variante estensionale dei sistemi "forbidding-enforcing" (ispirati dall'osservazione di processi chimici e applicati sia al calcolo DNA che al calcolo a membrane [37]), per modellare il processo di generazione spontanea di forme DNA tridimensionali. Un modello che nel contesto della teoria dei grafi potrebbe essere un punto di partenza per lo studio di una particolare classe di grafi che risulta interessante dal punto di vista biologico [74].

L'ultimo capitolo delinea alcune idee in corso di elaborazione, che ci si propone di sviluppare in seguito. In particolare, nelle prime sezioni sono presi in esame dei possibili algoritmi per silenziare geni specifici, mostrando qualche risultato iniziale di laboratorio, mentre nelle ultime ci si concentra su un modello a membrane del processo di rimarginazione del tessuto del ginocchio soggetto ad un taglio interno che può provocare artrite cronica.

10.1 Problemi e risultati principali

Nei capitoli centrali della tesi una variante della PCR viene proposta come strumento biotecnologico, sia per estrarre da un "pool" eterogeneo di stringhe DNA contenenti una data sottostringa, sia per generare librerie DNA, che per effettuare operazioni di mutagenesi.

Più nel dettaglio, sono stati affrontati i problemi con le seguenti specifiche:

1. Generare una libreria combinatorica di numeri binari di n-bit, codificati in sequenze $X_1, Y_1, \ldots, X_n, Y_n$ di DNA, ovvero produrre il seguente pool contenente 2^n diversi tipi di stringhe:
$$\{\alpha_1 \cdots \alpha_n \mid \alpha_i \in \{X_i, Y_i\},\ i = 1, \ldots, n\}.$$
2. Estrarre da un dato pool P tutte le stringhe che includono una data sottostringa γ, ovvero ottenere un pool che consiste di tutte le stringhe di P che contengono γ come sottostringa:
$$\{\alpha\gamma\beta \mid \alpha\gamma\beta \in P\}.$$
3. Effettuare mutagenesi (X, Y) su un pool P, ovvero sostituire nelle stringhe di P ogni eventuale occorrenza di X con un'occorrenza di Y, trasformando P nel seguente pool:
$$\{\alpha Y\beta \mid \alpha X\beta \in P\}.$$

Sono stati presentati alcuni bioalgoritmi, principalmente basati sull'azione della polimerasi, che rivelano notevoli vantaggi in termini di efficienza e realizzabilità rispetto ai metodi in letteratura. Il lavoro è stato supportato da analisi matematiche e da esperimenti di laboratorio. Questo ha incluso uno studio collegato ai sistemi di splicing ("splicing systems"), dato che la XPCR offre un nuovo modo di implementare una ricombinazione particolare che in natura è realizzata da pochi tipi di enzimi.

Abbiamo disegnato e implementato un algoritmo che produce una libreria DNA come un linguaggio testabile in modo strettamente locale ("strictly locally testable language") con un particolare sistema di splicing detto semplice ("simple splicing system") [164]. In letteratura, i sistemi di splicing partirono proprio da questo modello, mentre una caratterizzazione algebrica di questo tipo di sistemi splicing fu data in [103], la prima per qualsiasi classe di sistemi splicing. Abbiamo chiamato il nostro algoritmo *di ricombinazione quaternaria*, in quanto genera l'intera libreria a partire da quattro sequenze. Esso realizza in modo molto efficiente la produzione di librerie di cloni, che risultano fondamentali dal punto di vista biologico, per vari tipi di esperimenti di selezione in vitro: come individuazione di nuove sequenze DNA o di enzimi RNA, per effettuare cross-over di geni omologhi, o per produrre memorie DNA. Il risultato è incoraggiante, in quanto rivela una nuova capacità di manipolare DNA per ottenere ricombinazioni genetiche, ma per stabilire la scalabilità dell'algoritmo saranno necessari ulteriori risultati sperimentali. La tecnica XPCR è stata applicata anche ad algoritmi di estrazione e mutagenesi, che sono interessanti sia nelle computazioni DNA, dove l'operazione di estrazione effettuata dai protocolli standard è ancora problematica in quanto fortemente soggetta ad errori, e sia nella biologia, per esempio negli studi di mutazioni genetiche.

Infine, il modello per il self-assembly DNA proposto nel Capitolo 7 suggerisce nuove direzioni di ricerca, sia in teoria dei grafi che nel calcolo di forme DNA. Il problema lì affrontato può essere posto nei seguenti termini:

4. Dato un insieme $\mathcal{P}$ di cammini e di cicli, un insieme $\mathcal{F}$ di strutture "vietate" (forbidden), ed un insieme $\mathcal{E}$ di strutture "forzate" (enforced), caratterizzare i grafi inclusi nell'insieme $\mathcal{G}(\Gamma)$ per $\Gamma = (V, \mathcal{P}, E, \lambda, \mathcal{F}, \mathcal{E})$.

Essenzialmente abbiamo esteso l'idea dei sistemi forbidding-enforcing per caratterizzare una famiglia di grafi di interesse matematico e biologico. Nel caso di self-assembly DNA, il processo di evoluzione è stato descritto in modo naturale dall'aumento della cardinalità del "matching set", ovvero dell'insieme di archi tra vertici le cui etichette sono complementari. Per altri tipi di applicazioni il concetto di grafi forbidding-enforcing potrebbe dover essere adattato ai particolari processi di cui si vuole studiare l'evoluzione.

In conclusione, la tesi presenta sia modelli teorici di computazioni molecolari che nuovi algoritmi DNA, corrispondenti a procedure di una certa rilevanza biologica. In particolare, sono stati individuati specifici metodi di estrazione DNA [72], di ricombinazione DNA [77], e di mutagenesi [150].

Segue una lista dei principali risultati conseguiti.

Nel contesto dei modelli teorici, troviamo:

- un modello a membrana per il reclutamento selettivo dei leucociti [75, 43] - collaborazione con Vincenzo Manca, Dipartimento di Informatica, Università di Verona, Italia.
- una macchina a registri DNA - collaborazione con Maurice Margenstern, LITA, Università di Metz, Francia; lavoro ispirato dall'articolo [76] con Vincenzo Manca.
- un insieme di vincoli di tipo "forbidding-enforcing" su grafi, che descrivono processi di self-assembly [74, 73] - collaborazione con Nataša Jonoska, Dipartimento di Matematica, Università della Florida del Sud, Stati Uniti d'America.

Nel contesto delle speculazioni combinatoriche e algoritmiche su nuovi metodi di manipolazione DNA, troviamo (collaborazione con Vincenzo Manca):

- la tecnica XPCR
- un algoritmo di estrazione basato su XPCR [72]
- un algoritmo di mutagenesi basato su XPCR [150]

e come frutto di un lavoro maggiormente autonomo:

- un algoritmo di ricombinazione basato su XPCR [77, 68]
- un algoritmo di estrazione basato su operazioni enzimatiche [71]

Infine, sono stati condotti tre tipi di esperimenti per testare tutti gli algoritmi basati su XPCR - collaborazione con Vincenzo Manca, insieme a Cinzia Giagulli e Carlo Laudanna del Dipartimento di Patologia, Università di Verona, Italia.

10.2 Conclusione

Un aspetto saliente della tesi può essere ricercato nel lavoro di interazione, filtraggio, e traduzione, tra il lato teorico e quello sperimentale della ricerca presentata. Sebbene non la si possa considerare ricerca informatica "ortodossa", si direbbe un lavoro prezioso dal punto di vista scientifico, senz'altro esaltante, in quanto mette in luce una natura sperimentale delle stringhe (biologiche, modellate da quelle formali) e degli algoritmi. Il Capitolo 4 è un semplice esempio di come speculazioni del tutto teoriche, quali quei risultati di combinatorica, possano essere da un lato motivati e ispirati da, ma dall'altro anche molto utili e significativi per, un esperimento di calcolo col DNA in un laboratorio di biologia molecolare.

Abbandonando il lavoro di questa tesi e andando a quello che la comunità internazionale sta portando avanti col calcolo DNA, è doveroso far presente che finora ci sono stati pochi progressi nel dimostrare concretamente che le computazioni DNA sono capaci di risolvere problemi che non possano essere risolti banalmente da computer convenzionali. Rimane dunque un problema aperto quello di decidere se il calcolo col DNA sarà mai "utile" all'informatica da un punto di vista pratico. Ma la scienza ben supera le restrizioni dettate dai sistemi implementabili oggi o domani, ponendosi come obiettivo quello di studiare computer plausibili e i modelli di calcolo da essi ispirati. D'altra parte, il progresso nella tecnologia, che sia fisica o biochimica, è talmente rapido e imprevedibile che sarebbe anche poco saggio limitare i propri studi alle conoscenze e agli strumenti utilizzabili oggi o domani.