Assisted Design of DNA Computing Systems: the DNA Toolbox and Beyond

(DNAコンピューティングシステムの設計支援：
DNAツールボックスとその拡張)

by

Aubert Nathanael Yann Ivan
オベル　ナタナエル ヤン　イバン

A Doctor Thesis
博士論文

Submitted to
the Graduate School of the University of Tokyo
on December 13, 2013
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Information Science and
Technology
in Computer Science

Thesis Supervisor: Masami Hagiya　萩谷　昌己
Professor of Computer Science

Thesis Cosupervisor: Yannick Rondelez
Professor of Biochemistry

ABSTRACT

In the recent years DNA computing has shown promising results with potential applications in a wide range of fields, such as medicine and smart material engineering. DNA computing leverages the well understood biochemistry of DNA to encode calculus and perform computation, with the goal of enabling computation in inconvenient places, such as inside the human body. However, most ideas often stay at the stage of proof of concept, without making it to the marketable product level.

The reason for this is not a lack of interest from designer, but lies instead in the fact that such products have a structural complexity that is beyond the tools currently available in the field. Even though interactions such as DNA-DNA or DNA-enzyme are reliably predictable, they cause counter-intuitive behaviors, such as Michaelis-Menten enzyme saturation. For this reason, even simple systems may require a long trial-and-error design process.

The goal of this thesis was to develop means to ease the design process of such systems. This relies mainly on three complementary approaches: module design, development of specific computer assisted design (CAD) tools, and search of interesting patterns evolved through artificial evolutionary algorithms. Those different aspects are each used to simplify a particular aspect of the design process.

The first step was to create a new module that could be easily integrated to popular DNA computing paradigms. The role of this module is to perform a delay or join (in the traditional computer science meaning) operation. Indeed, one major design problem is that intermediary reaction products can react with any other part of a system, not only their intended target. While careful sequence design can avoid unwanted interactions up to a certain point, it still leaves the problem of leaks: during a given computation step, the wrong output may be present as a temporary product, before the correct one is finally released. If the incorrect output is allowed to interact with downstream elements of the system, errors will propagate. This problem is akin to concurrency problems in computer science, and can be resolved using the delay module by ensuring that a given computation is finished before allowing the next step to happen. More specifically, the delay module relies on timer strands capturing the targeted sequence and being then catalytically consumed by a DNA delay gate. Timer strands are present in excess with respect to their target, and duplex are designed to be only consumed once all single-stranded timers are gone, ensuring the delay. This approach was confirmed both by simulation and experiment.

The second step was to develop an efficient simulator to help the designer prototype systems. Indeed, while actual experimentation is necessary to check the correctness of a system's behavior, the cost in time and resources prevents its use as a convenient way to design. Instead, simulation allows the user to refine the system and check for flaws. We targeted the DNA toolbox, a set of three modules (activation, inhibition and autocatalysis) used for bottom-up design. Its simple modules allow a straightforward representation of systems as a graph. Moreover, those modules can be easily derived into a mathematical model close to their actual *in-vitro* behavior. This means that systems designed with this CAD software have reasonable chances of behaving in the same fashion in an actual implementation. The software also offers to export designed systems in the Synthetic Biology Mark-up Language format, allowing the user to use other design tools, such as Copasi.

The third step was to take the opposite design approach: since it is hard to come up with a structure performing a given task, it is instead possible to look at the possible behaviors that arise from small structures. To direct this search, we used an *in silico* evolutionary approach. Systems were asked to "play" a simple game of rock-paper-scissors, with a fitness based on their success rates. By performing an analysis of the evolution of well-performing systems, it was possible to highlight multiple substructures with specialized roles. While most of those building blocks were specific to the problem at hand, they showed the interest of this approach.

論文要旨

近年、DNA コンピューティングに関する研究は、創薬や知的材料など、幅広い分野への応用の可能性を示唆する有望な結果を生み出している。DNA コンピューティングは、DNA に関する既知の確立した生化学を活用して、計算系を分子系にエンコードして計算を実行し、最終的には、人間の体内のように、従来のコンピュータにとっては不便な場所において、計算（情報処理）を可能にすることを目標としている。しかしながら、大半のアイディアは概念実証の段階にとどまっており、製品化の段階には達していない。

その理由は、開発者に製品化への意欲が欠落しているからではなく、そのような製品の構造が、この分野で現在利用可能なツールによって取り扱いが可能なレベルよりもはるかに複雑だからだ。DNA 同士もしくは DNA と酵素の個々のインタラクションは十分予想可能であるが、それらは複合して、ミカエリスメンテンの酵素飽和反応にみられるような反直感的な挙動を引き起こす。このため、非常に単純なシステムでも、その設計には多くの試行錯誤を要する可能性がある。

本論文の目標は、上述のような DNA コンピューティングを行うシステムを設計する過程を効率化する方法を開発することである。この目標を達成するために 3 つの相補的なアプローチをとる：モジュールの設計、計算機支援による設計 (CAD) ツールの開発、および、人工進化計算アルゴリズムによる新奇パターンの探索である。これらの異なるアプローチは、それぞれ、DNA コンピューティングを行うシステムの設計過程において、特定の側面を効率化するために用いられる。

本研究の最初の段階は、一般的な DNA コンピューティングのパラダイムに容易に取り込める新しいモジュールを創ることである。このモジュールの役割は、(伝統的な計算機科学の意味で) 遅延もしくはジョインの演算を行うことである。DNA コンピューティングを行うシステムの主要な設計問題の一つは、中間生成物がシステムの標的以外の他の部分と反応することである。注意深く塩基配列を設計することにより、望まれない反応をある程度防ぐことができるが、リークが生じるという問題は残されている。すなわち、特定の計算段階の途中で、最終的に正しい出力が生成される前に、正しくない出力が一時的な産物として残留する。正しくない出力がシステムの下流の要素と反応することが許されれば、エラーは伝播する。この問題は計算機科学で見られる競合問題と同種であり、遅延モジュールを用いて、次の計算段階が起こる前に現在の計算段階を終了させることで解決され得る。より詳細に述べると、遅延モジュールは、タイマー鎖が標的の鎖を捕らえ、さらにその鎖が DNA 遅延ゲートによって触媒的に消費されることにより機能する。タイマー鎖はその標的に対して過剰量存在し、全ての一本鎖タイマーがなくなったときにのみ、二重鎖が利用可能になるように設計されており、遅延を確実なものとする。このアプローチはシミュレーションと実験により実証された。

次の段階は、プロトタイプシステムの設計を補助するための効率的なシミュレータを開発することである。実際の実験はシステムの動作の正しさを確認するために必要であるが、多大な時間とコストがかかるため、システム設計において簡便な動作確認手段にはならない。一方、シミュレーションはシステムの性能向上とその動作確認を可能にする。我々は、ボトムアップ設計に用いられている DNA ツールボックス (activation：活性、inhibition：阻害、autocatalysis：自己触媒の 3 つから構成される) に注目した。その単純なモジュールは、グラフとしてシステムを直接的に表現することを可能にする。さらに、それらのモ

ジュールに対して、実際の試験管内の挙動に近い数学的なモデルを与えることができる。したがって、この CAD ソフトウェアを用いて設計されたシステムは、実際の実装においてもシミュレーションと同様の挙動を示すことが予想される。また、このソフトウェアは、設計されたシステムを Biology Mark-up Language の形式で出力する機能を有しているので、ユーザが他の設計ツール、たとえば Copasi を使用することを可能にする。

3 つ目の段階では、以上と反対の設計アプローチをとる。すなわち、所望の機能を持つ構造を思いつくのが困難であるときも、小さな構造を組み合わせることで生み出される挙動を調べることは可能である。このような探索を行うために、我々は計算機を用いた進化のアプローチを採用した。システムは互いにじゃんけんを行い、じゃんけんの勝率に基づく適合度によって進化する。じゃんけんに強いシステムが出現する過程を解析することにより、特定の役割を持つ複数の部分構造を特定することができた。これらの部分構造ブロックのうち、大半はこの問題に特殊なものであったが、本アプローチの有効性を示している。

Acknowledgement

First of all I want to express all my love and affection for my familly and friends back in France. It has been hard to stay in touch because of this PhD, so it is only fair that your turn comes first.

I want to thank Professor Masami Hagiya, my PhD supervisor, for having believed in my work and supporting me despite his busy schedule. I also want to thank Professor Yannick Rondelez, my PhD co-supervisor, for giving me the opportunuity to work in his group, giving me a (hard) push when I needed one, and being an overall awesome human being. He also had to endure correcting literaly hundreds of pages of my French English throughout the duration of my PhD.

I greatly appreciated the various comments from my jury members, which helped polish this thesis to its current state.

In no particular order, this work is also dedicated to my colleagues and friends Ibuki Kawamata, Fumiaki Tanaka, Adrien Padirac, Anthony Genot, and Alexandre Baccouche, who taught me a lot about DNA computing and experimental work; to Takako Kato who kept me sane, healthy and motivated; to Christophe Provin, Pierre Alain, Dennis "Dede" Damiron and the rest of the LIMMS; to Olaf Witkowsky, Julien Hubert, Jose Alvarez, Paulo Silva and all my other friends from Tokyo, for making my everyday life interesting and crazy.

I also want to sincerely thank Nicolas Bredeche, Andre Estevez-Torres and Alexis Vlandas for fruitful discussions and collaborations.

I would like to mention all the DACCAD beta testers who took time to provide detailed bug reports.

Among the less(er) formal acknowledgements, I want to thank Tharol Hunt and Dan Shive, webcomic artists. We never met, nor even communicated, but your work definitely saw me through the worst time of my thesis. Many thanks, finally, to the University of Tokyo for lending me a computer and giving me a nice workspace. My chair for its unyielding support.

I wouldn't have achieved a tenth of what I did without them.

Contents

Chapter 1

Molecular computing and DNA-based systems

1.1 Introduction

For the past decades, the use of personal computers has become so ubiquitous that the original meaning of the word (a person carrying on calculation) has effectively fallen out of usage. In popular culture, the image of a computer is that of a general purpose, programmable electronical device, with standardized inputs and outputs (monitor, mouse, keyboard, USB ports). However, it was not so long ago that researchers were stuggling with a then new technology, trying to find the best paradigm for their machines. Over time, many avenues, such as void tubes and mercury canals, were explored until the current digital logic gate circuitry have been perfected. Moreover, even though the technology became more and more standardized, implementations were far from homogeneous. Even before the famed Moore law[1] started to show signs that it could not keep its trend without changes to the monolitic processor design, multiple alternatives branched out (vector programming, graphics processing units, among others). Some of those found niche applications, while many other died, in a striking analogy with the evolution of species and the survival of the fittest.

However, virtually any modification of a given environment can be seen as a computation being carried out, to the point that in his science-fiction novel "Hitchhicker's guide to the galaxy", Douglas Adams presents planet Earth as a gigantic computer [1]. Alan Turing formalized the idea of calculation [2] and introduced the Turing machine, a model for universal computation. This means that any calculation that can be computed can be encoded as a Turing machine

[1]The Moore law predicts that computational power (represented as the density of transistors) will increase exponentially over time. The average rate was to double the clock speed of computers every two years.

performing operations following a particular set of rules. Alternative computation paradigms are often referred to as unconventional computing. The main idea is to find something that is Turing universal (i.e. has at least the same computational power as a Turing machine, see glossary) while having interesting properties, such as new fields of application like nanorobotics or distributed systems. While some of these alternative approaches are well-known, like cellular automata or quantum computers, some are more exotic, like ant computing [3] or swarm intelligence [4].

1.2 Toward molecular programming

A prominent alternative approach is that of molecular programming, where chemical species are defined to encode data, while chemical reactions represent computational operations. The result of a given calculation can then be represented as the transient concentration of some species [5, 6, 7], the steady state of the system [8, 9, 10, 11, 12] or even the presence of some particular species, regardless of their actual concentration [13, 14]. To perform complex calculus, such chemical reactions are cascaded to form what is known as Chemical Reaction Networks (CRN) [15]. An interesting feature is that CRNs have been proved to have Turing universality [8, 16]. Moreover, multiple CRNs computing the square root of an input, for instance, have been shown [17, 10], demonstrating the practicality of this computation approach.

While theoretical power and programatical approaches of a given paradigm have an incontestable interest, possibly giving new insights for algorithms (as with quantum computing [18]), such paradigm will lack impact without a proper “real life” implementation. Molecular programming is no exception to this rule: artificial chemistries [19], paradigms using the formalism of chemistry to propose particular reaction sets for specific objectives, have little application if they cannot rely on a wet lab implementation, but taking this step leads to many problems. While proofs of concept might be achieved with specific set of reactions [20, 21], it is, in the general case hard or even impossible to control such reactions. That is, many unexpected side reactions can occur among species in even trace quantities, which, when cascaded with the reaction network, might lead to a completely arbitrary state of the system that carries no meaning anymore (which can be described as “tar”).

However, examples of complex reaction networks are present in any living cells: protein expressions, gene silencing and others form tangled webs of interactions. Moreover, those networks bear a surprisingly close resemblance to call graphs from computer software (Figure 1.1).

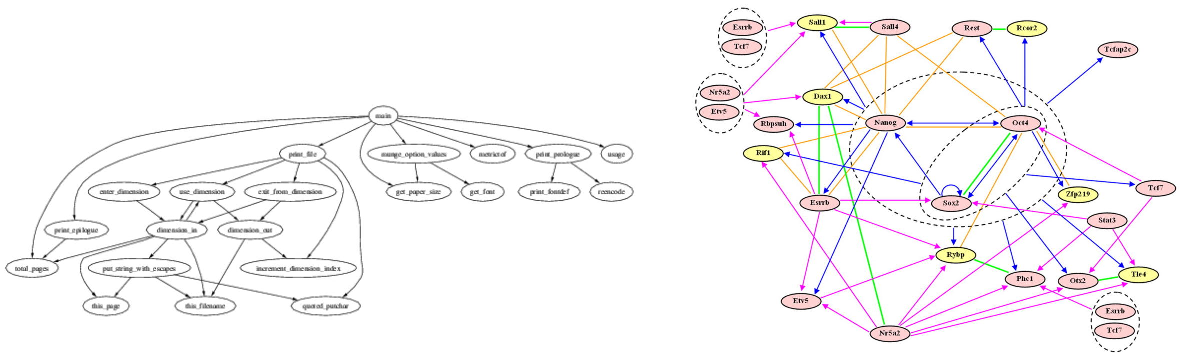


Figure 1.1: Comparison between a small program static call graph and a gene regulatory network. Left: Call graph mapped using Egypt [22]. Right: gene regulatory network in the mouse embryo [23]. While the call graph seems more ordered, one should bear in mind that the displayed program is very simple and only displays text. The call graph of a medium-sized program would be completely unreadable. Also, note that both graph contains loops.

This resemblance is not fortuitous, as the cell is indeed carrying out computations leading to impressive feats such as cell growth and division as well as complex reading of its environment, such as the identification of foreign entities by the immune system. It is then obvious that mechanisms have been evolved through the ages to keep chaos at bay, since all we could expect from a random mix of chemicals is the "tar" mentioned before. In particular, DeoxyriboNucleic Acid (DNA) molecules and their close relatives RiboNucleic Acid (RNA) molecules, as a side-effect of their role as the carrier of genetic information, have a comparatively precise chemistry. Indeed, reactions among DNA molecules will never lead to the creation of non-DNA products, for instance, and are predictable due to a phenomenon called Watson-Crick pairing [24] which governs the way DNA molecules will interact with each other. Moreover, the number of possible reactions, while large, is fairly restricted (see next Section). For this reason, DNA molecules have been gathering a growing attention for the past twenty years as a promising material for molecular computing. Adleman first proposed the use of DNA computing in 1994, using the interactions between specifically crafted DNA molecules to solve the traveling salesman's problem [13]. This is not limited to DNA-only interactions, as enzymes have evolved to have specific interactions with DNA and RNA, even going as far as including proof-reading steps, and represent a huge spectrum of possibilities. Moreover, those possibilities are getting known better and better, first as part of studies on biology, then as full-fledged elements of interest. Thanks to its high controlability, DNA has proven capable of creating structures [25, 26, 27, 28, 29], actuators [30, 31, 32, 33], walkers [34, 35, 36], sensors (called aptamers) [37], storage [38, 39] and to carry on computation [6, 40, 10, 41, 12, 42, 43], making it a molecule of choice for a wide range of applications. In particular, it is possible to put a DNA-based CRN in bulk in a

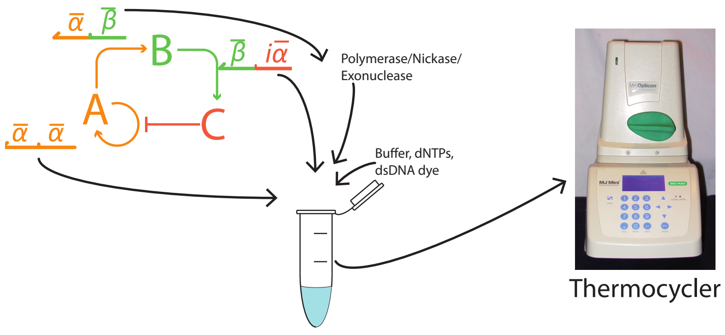

$\bar{\alpha}$ $\bar{\beta}$
B
$\bar{\beta}$ $i\bar{\alpha}$
A
C
$\bar{\alpha}$ $\bar{\alpha}$
Polymerase/Nickase/ Exonuclease
Buffer, dNTPs, dsDNA dye
Thermocycler

Figure 1.2: Mixing DNA sequences (arrows), enzymes and buffer solution together in test tube and observing its behavior by fluorescence in a PCR machine.

closed system and observe it (Figure 1.2). Its biological origin makes it particularly suited to interface with living organisms, to perform either smart drug delivery [44], medical diagnostic [45] or gene expression monitoring [46]. Additionally, combining its various capabilities also makes DNA a valid possibility to develop nanomachines and smart materials [47, 48, 49].

Another strong point of DNA is its availability and price [50]. Many companies offer to synthesize arbitrary DNA sequences, with or without modifications, and then ship them. The whole process takes between one day and a month, depending on the amount of artificial modifications requested, which guarantees a fast turn-over of the experiment-redesign-order sequence.

1.3 A crash course in DNA computing: abstraction and possible operations

DNA is a fairly complex molecule in its own right if we consider the number of atoms required to create its structure (Figure 1.3, left), but behave in a way robust enough to allow us to abstract most of this complexity away. While it is possible to model each and every single atom in DNA [51, 52], such studies (as well as experiments) show that it is safe to scale down the amount of details taken into account; Figure 1.3 shows multiple possible levels of abstraction. A DNA *strand* is composed of a backbone of sugar and phosphates on top of which nucleotides (A,T,G,C) are attached. The nucleotide structure is such that A (respectively C) forms a bond with T (respectively G) almost exclusively [24]. This level of details is used to define *coarse-grained* models of DNA, such as oxDNA [53, 54]. Those models try to optimize the amount of retained details (structural or otherwise) while having a computational cost light enough to simulate non-trivial systems. The next level of abstraction removes most structural information of DNA strands. At this stage, it is important to note that DNA strands are oriented, with two different ends denoted $3'$ and $5'$ (those numbers are references to the structure of the backbone

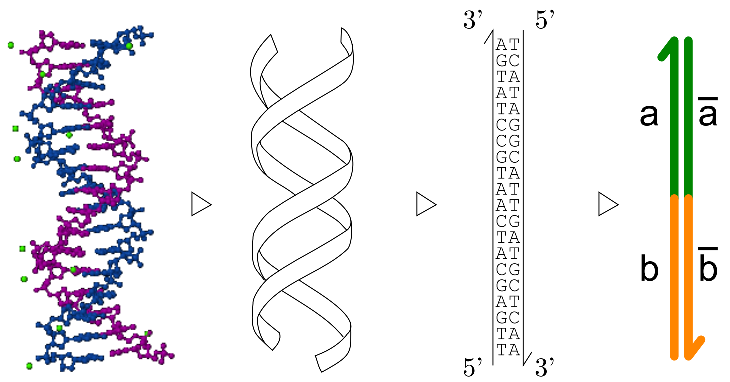


Figure 1.3: Four levels of abstraction. Left: the actual atoms making up the molecule are taken into account, but considered as solid spheres while the covalent bonds are represented as sticks. This DNA molecule comes from the Protein Data Bank database, reference 3bse. Center left: the atoms are abstracted, leaving only the general shape. This particular level is mostly useful for DNA-based structures, such as in DNA origami [26] or DNA lego [55] where the actual number of turns can compromise the final shape. DNA curvature can also be used to produce curved structures [56] or altered alternatively to produce actuators [30]. Center right: the shape is also abstracted and DNA strands are then represented as arrows going from their $5'$ to $3'$ end. This level applies Watson-Crick base-paring [24] and is mostly used to confirm the absence of unexpected crosstalks in a given system. Right: final abstraction level. Nucleotide strings are grouped in domains based on their role in the system and represented by a specific letter. This is thus a logical abstraction rather than a physical one. Color is often added, like in this example, to improve readability.

at the strand end-points), thus leading to representing strands as arrows (Figure 1.3, center right). DNA strands can be further segmented into *sequences*, also named *domains*[2]. A DNA *sequence* a is said to be complementary to a sequence $\bar{a}$ if and only if each nucleotide in a can form a bond with the corresponding nucleotide in $\bar{a}$ when the two sequences are aligned.

When two sequences are complementary, they can bind to form the well-known double-helix in a process called *hybridization*. Note that the two DNA strands will hybridize head-to-tail, so that the complementary of ATGC is GCAT, not TACG. When two strands of DNA are hybridized, the resulting molecule is called *double-stranded* DNA. Note that a DNA strand can hybridize to itself if it contains complementary domains, forming a structure called hairpin (Figure 1.4, b). The opposite reaction, when two hybridized sequences separate, is known as *denaturation*, melting or dehybridization[3]. Depending on the length of the sequences (longer sequences are more stable), their nucleotides (G-C pairs are more stable than A-T), the potential presence of a *mismatch* between nucleotides, the experimental conditions (the amount of magnesium or other salts, for instance, changes stability) and other factors, the chances for a given sequence to be double-stranded will change; the more stable, the more chances it has to be

[2]While those two words are mostly interchangeable, sequence refers more to the physical, actual, succession of nucleotides while domain is a logical abstraction.

[3]The name changes based on the method used to accomplish the process.

double-stranded. It is possible to evaluate the temperature at which a given DNA molecule is double-stranded only 50% of the time due to thermal agitation [57], called melting temperature. This gives us valuable information on interactivity among DNA molecules. For instance, long sequences (20 bases or more) are stable enough to be considered to not denature spontaneously at room temperature, so fully double-stranded DNA of this size can be considered inert. The kinetics of such reactions have been well studied, and many direct measurement methods exist (see for instance Nelson and Tinoco [58]).

The last DNA-based reaction we consider is called *strand displacement*. In this case, two different strands A and B are competing for a specific domain on a third DNA molecule. In the case where A is already hybridized with the target sequence, but B has a nearby *toehold* (see Figure 1.4, c), B can eventually take the place of A, based on the progressive exchange of base-pair bonds. If A does not have a toehold itself, the reaction is irreversible. On the other hand, if A also has access to a toehold, it will be able to initiate a strand displacement in the other direction, meaning that the resulting molecule will have its domain shared among A and B in an ever changing fraction. If the toeholds are small, this process can lead to a cascade of toehold-mediated strand displacement, called toehold-exchange [59].

While thermodynamics governs the possible states during strand displacement, it is also interesting to consider the kinetics of this reaction, especially for real life applications. The closer to the displaced sequence [60] and the longer [61] the toehold, the faster strand-displacement is. In general, intermediary steps (the random walk between the initial and final step) are abstracted in the overall reaction speed, since they do not carry meaning[4].

Other DNA- or RNA-based molecular reactions exist, such as DNAzymes (specific DNA molecules performing structural modifications on other molecules, like in [35, 36]) and aptamers (DNA structures that can recognize and attach to particular targets, such as molecules [62], proteins [63], and so on), but are beyond the scope of the present work. Integration of such processes could be a relevant future extension of the DNA toolbox, the DNA computing paradigm that we use in this thesis.

Hybridization and denaturation alone, if used correctly, have proved to have the same computational power as a Turing-machine, although with limitations, following schemes such as sticker-based computation [64] (requires external operations) or DNA-tiles [9] (leads to the growth of large structures, and thus is not scalable). The addition of the strand displacement

[4]They might still initiate unexpected reaction cascades if a partially freed DNA strand starts interacting with other parts of the system even though it was not supposed to detach. This can happen, for instance, when toeholds are shared among multiple strand displacement gates.

operation allowed Winfree and co-workers to elaborate a computation paradigm that proved to be cascadable, as the output of one strand-displacement can initiate another [8, 59, 6]. Qian *et al.* built upon this mechanism to design a multiple stack machine (a Turing machine equivalent) [65]. Qian and Winfree also combined multiple steps of this mechanism to form a higher-level modular structure called the seesaw gate (Figure 1.5). The seesaw gate is a particular sequence of hybridization and strand-displacement that takes an input and release an output without signal loss in the case its level is higher than a programmable threshold [66]. They used this seesaw gate to implement both complex logic circuits [10] and neuron networks [41]. Soloveichik *et al.* have also shown that arbitrary CRN can be implemented using this approach [40].

The second type of possible operations on DNA is based on enzymatic reactions. Enzymes are catalysts that perform various tasks on single-stranded or double-stranded DNA. For each possible operation, it should be noted that a very large number of enzymes found in nature, as well as a fair amount of artificial one exist. The DNA toolbox [67], the paradigm we use in this work, relies on three specific enzymes: the *polymerase* extends a double-stranded DNA strand (called primer) in the $3'$ direction so that it matches the opposite strand using dNTPs (roughly speaking, the building blocks of DNA); the *nicking enzyme* recognizes a specific double-stranded sequence, and cuts ("nicks") the backbone of one of the two strands at a specific position; the *exonuclease* attaches to any single-stranded $5'$ end and processively degrades the strand. The list of all the actions we consider in this manuscript, including such enzymatic mechanisms, is summarized in Figure 1.4. Among other notable enzymes that are not used in the toolbox, we should mention the ligase, which performs the reverse operation of the nicking enzyme, and the restriction enzyme, which cuts the backbone of both DNA strands in an enzyme-specific place. Apart from the DNA toolbox, other systems use enzymatic operations. Adleman used ligase to form the solutions to the traveling salesman's problem [13], restrictase and ligase have been combined to implement finite-state machines [68], and the polymerase can be used both for computation [69] or movement [70].

Finally, it is interesting to note that there has been a recent effort to produce "DNA-like" molecules with various properties. Indeed, since the principles of DNA computing are generic, it makes sense to improve the "wetware" implementing them. There are two complementary approaches: creating different backbones and creating new nucleotides, grouped under the name XNA (xeno NA). DNA-like molecules with exotic backbones can be synthetically created and come in many flavors [71, 72, 73, 74]. Of particular interest, we can cite L-DNA [75], LNA [74] and TNA[73]. L-DNA (left-handed DNA) has a backbone mirroring that of the naturally

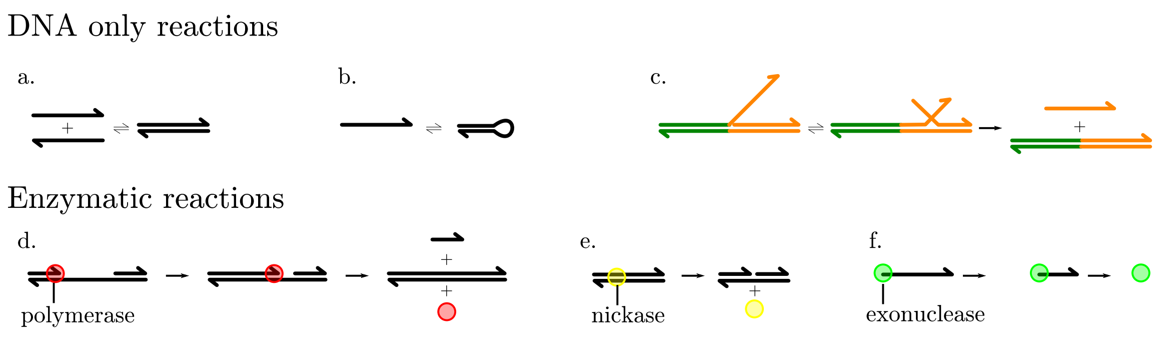


Figure 1.4: Main operations possible on DNA. Double harpoons represent reversible reactions, while simple arrows represent irreversible ones. **a.** hybridization (left to right) and denaturation (right to left). **c.** Strand-displacement. **d.** polymerization of a DNA strand, showing here the strand-displacement activity of the polymerase. **e.** nicking. **f.** hydrolysis by the exonuclease.

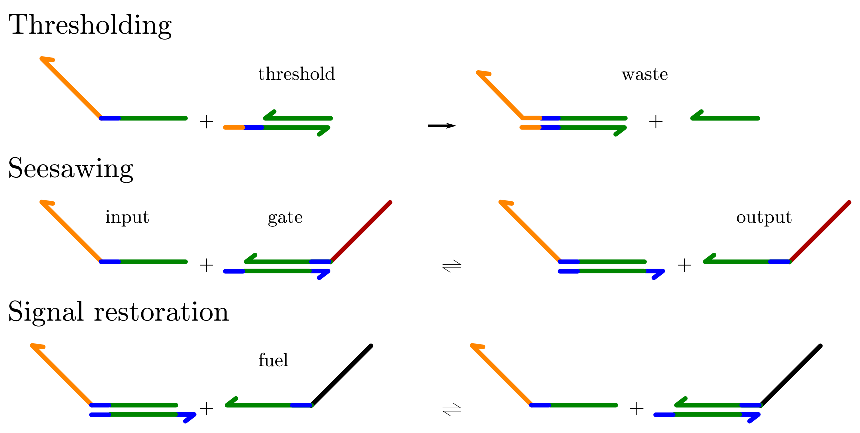


Figure 1.5: The three parts of a seesaw gate. Thresholding: the input is first wasted by a given amount of threshold species. The input interacts preferentially with the threshold species due to the longer toehold. Seesawing: the input and output compete for a DNA substrate through toehold exchange. Signal restoration: fuel takes the place of the output on the gate, allowing the input to displace the totality of output, even after being partially wasted by the threshold species. Note that the fuel does not need to have the black tail represented here. A more detailed explaination is given by Qian *et al.* [10].

occurring DNA molecule, so that it behaves normally with other L-DNA molecules, but cannot hybridize with regular DNA. Similarly, the mirrored double helix prevents non-modified enzymes to interact with this molecule. However, standard (right-handed) DNA and L-DNA molecules can be joined together by covalent bonds, opening the door to a wide range of new structural possibilities (see for instance its application to the creation of microarrays [76] or of arbitrary shapes for DNA structures [29]). LNA (Locked NA) is DNA molecule with a slightly modified backbone which makes its double-stranded conformation orders of magnitude stronger, while having a higher mismatch discrimination [77]. The LNA modification can be (and usually is) applied to specific nucleotides in a given DNA strand (creating what is called an LNA-DNA chimera), allowing denaturation to still take place while increasing the stability and specificity of the modified sequences. TNA (Threose NA), a molecule with a threose sugar instead of ribose in its backbone, is interesting for the efforts that were done to create an artificial polymerase capable of transcripting it into DNA and back, making TNA stand out as one of the only modified-backbone DNA analogues with enzymatic compatibility.

Creation of new nucleotides pairs, beside AT and GC, is also noteworthy, as it extends the alphabet that can be used to encode sequences, reducing the amount of potential crosstalk (see next Section). The drawback, like with artificial backbones, is that enzymes are usually not compatible with them. Besides the obvious limitation of DNA computing operations, there is also a time and cost problem, as the creation of modified DNA sequence has then to rely on non-enzymatic chemistry, which has a tremendously lower yield. However, recent years have seen the apparition of both newly engineered enzymes [78] and compatible nucleotides [79, 80].

Without going to such extremities, simple nucleotides modifications, such as attaching molecules (biotin, fluorophores, and so on) at the end of a DNA molecule, should also be mentioned, as they are common and add multiple functions to DNA, used for purpose as various as monitoring, diffusion drag or interfacing with other molecular compounds.

1.4 Limitations and workarounds

Twenty years ago, Adleman predicted that DNA computing might be able to compete with electronic-based computing [13]. However, it has become apparent that it will not be the case: while DNA computing has the advantage of offering massively parallel operations, it suffers from many drawbacks.

- Reaction time: DNA-based reactions, with or without enzymes, are orders of magnitude slower than electronic-based computation. This problem is directly balanced, however, by the massively parallel computational power of DNA and the potential applications of DNA we mentioned. Parallelism means that virtually all combinations of a problem[5] can be solved at the same time, which was proposed to solve the SAT problem or its 3-SAT variant [82, 14, 83], among others. Another reason why reaction time is not as limiting as in electronic-based devices is the environment in which DNA computing can be used. It has the ability to work *in-vitro* (in a test tube) or *in-vivo* (in a living organism, a cell, . . .), where no silicon computer can work. In such context, speed might not be a limit: low computational power is enough for applications such as drug delivery, symptoms assessment or simple nanomachines composing smart materials. Ants [3] and termites [84] showed that the combination of limited computational power distributed over a large number is enough to create impressive achievements.

- Sequence space: the DNA addressing space is limited. Contrary to digital computers where a pointer will be able to discriminate between its target and nearby memory addresses, DNA sequences that are too similar will be able to interact, albeit at a much slower rate, which might still yield unexpected results. For this reason, the sequence design step is critical when implementing an *in-vitro* or *in-vivo* systems. This problem can be worked around either by using indirect addressing (checking the correctness of the sequence through strand-displacement or mechanisms such as meta-DNA [85]) or artificial nucleotides. The delay gate we propose (Chapter 3) can also help by inactivating targeted parts of the system, limiting such potential crosstalks.

- Lack of interface: it is very hard to interact with DNA computing systems, as the technology is still lagging in this area. For the most part, systems have to be prepared painstakingly by a human in a wet lab and any input has to be injected manually. The reading of outputs is either done by measuring fluorescence in real time in a repurposed PCR (Polymerase Chain Reaction) machine (see Figure 1.2) or by a process called electrophoresis, which requires to wait for the end of the monitored reaction and takes at least an additional half a day. Moreover, while fluorescence is measured in real time, the number of different colors that can be monitored at the same time is limited by light wave-

[5]There is a physical limitation due to the maximum concentration of DNA that can be fitted in a test tube. Single-stranded DNA at a concentration higher than 130 mg ml^{-1} will spontaneously start to crystallize [81], which will change the reaction possibilities. This concentration is equivalent, for 100 base-pair long single-stranded DNA molecules in a typical 20 mul reaction volume, to 5×10^{16} molecules, which is a small number when it comes to combinatorial problems.

length interferences to a very small number of possibilities, usually around four or five (depending on the machine precision). This leads to a difficult debugging of experimental systems, mostly based on trying to reverse-engineer the observed transient fluorescence and then design additional test experiments to validate those new hypothesis (see Chapter 6).

  This situation is changing, as automated mixing of DNA systems [55] as well as microfluidic [86, 87] and electrical [88] controls are starting to appear.

- Non-linear behavior: additionally to the limitation of sequence space, the fact that DNA computing systems behave in complex ways is a two-edged sword. On the one hand, it guarantees the richness of possible actions. On the other hand, designing such system is a nightmare of trial-and-errors. Enzymatic saturation, for instance, can change completely the dynamics of a system, for the worse [89] or for the best [42].

1.5 Goal and contributions of this thesis

The ideal of designing a system from its inception to the actual nucleotide sequences and having it immediately work in a test tube (or any potential environment) is still far off. However, there have been a lot of efforts toward making each step as simple as possible.

In this thesis, we focus on easing the design process of DNA computing systems, with the ultimate goal of allowing a fast development of *in vitro* systems. We believe that, to create systems beyond the proof of concept, it is important to alleviate as much the design process as possible. This thesis will thus introduce a number of tools for the design of DNA-based systems and presents how to use them.

To reach this goal, we explored three complementary directions, common in technological development: design robust modular building blocks, make the creation process of DNA systems faster by creating Computer-Assisted Design (CAD) tools and use automated optimization to help the user find novel solutions to hard problems. Robust blocks mean that they will behave predictably, so that it will be possible to model them and create reliable simulators. At the same time, modularity means that such blocks can be freely combined, which is an important condition both to scale up systems and to conduct automatic system generation. Finally, the automated optimization of parameters, either local (such as DNA species concentration in the system) or global (the reaction network is modified as well), helps solving problems where no simple solutions exist. Figure 1.6 shows the interaction between those aspects.

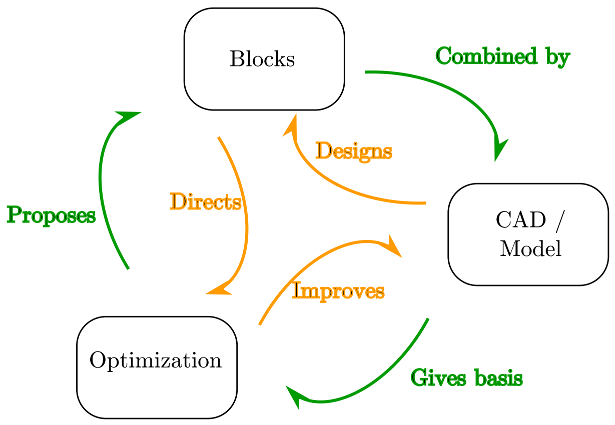


Figure 1.6: The three aspects we explore. Reliable building blocks are needed to create a model that is high-level enough to design large scale systems. At the same time, CAD software can create a library of robust basic patterns that can be reused across multiple systems. Systems thus modeled can be optimized by various algorithms, depending on the required scope of optimization (either local or global). Conversely, those optimized systems can be tinkered with, through the graphical interface of our CAD application. Finally, optimized systems may display interesting patterns that can be added to the block library. Such patterns can in turn direct optimization algorithms (that is, the search will be biased toward using such patterns).

The contributions of this thesis can then be classified among those three axis:

- New building blocks:
  - Hardware: the delay gate (Chapter 3), a new "hardcoded" mechanism that can be combined with other modules, notably from the DNA toolbox.
  - Software: new modular patterns based on the DNA toolbox that can be reused in a variety of applications (Chapter 7).
- Computer-Assisted Design:
  - We developed DACCAD (DNA Artificial Circuits Computer-Assisted Design, Chapters 4 and 5), a graphical user interface used to create quickly DNA toolbox systems. Those systems are simulated using our own detailed model of the mechanisms underlying the modules of the DNA toolbox (Chapter 2).
  - We created an interface to combine multiple paradigms. Mathematica can be used to combine a simple model of the DNA toolbox with the delay gate (Chapter 3) or other systems, such as DSD systems. DACCAD also exports SBML files [90, 91] that can be read (and edited) by a large range of programs.
  - We also ventured toward debbuging of DNA toolbox systems (Chapter 6).
- Optimization:

- local: improvement of a given behavior, using CMA-ES [92], a state-of-the-art optimization algorithm (Chapter 4).
- global: creation of a system from scratch using bioNEAT, an Evolutionary Algorithm designed to evolve systems from the DNA toolbox (Chapter 7).

We can note that, even though those three axis are biased toward theoretical approaches, they are grounded in reality by being based on close collaboration with experimentalists. We mostly build on the DNA toolbox, so that reliable experimental data were already available. When we created the delay gate, a totally new module, such data were obviously lacking, and we performed the relevant experiments to check feasibility. Moreover, the last level of implementation of the systems we explore, the actual sequence design, has already been made tremendously simpler and faster by tools such as NUPACK [93] and DINAmelt [57, 94].

1.6 Assisting the process of molecular programming: outline

After presenting in Chapter 2 the DNA toolbox, on which we focused our efforts, we will describe in Chapter 3 a potential new building block, the delay gate. When considering building block design, it is much easier to have a limited set of simple and combinable operations rather than many complex ones. This increases the capacity of the designer to rationally design a system from the bottom up and allow the creation of specialized compilers. In electronics, this leads to the creation of the current RISC (Reduced Instruction Set Computing) processors. In DNA computing, this leads to the development of many simple sets of operations, like DNA Strand-Displacement [61], genelets [95] and the DNA toolbox [67]. However, "limited" does not have to mean "minimalist", as cascading too many basic blocks has a time cost (see for instance Qian and Winfree [10]), as well as undesirable side effects such as the load effect [95]. The delay gate is a module that could benefit greatly from a "hardcoded" implementation, with a dedicated mechanism. Moreover, the delay gate has the advantage of being compatible with all the mentioned sets, but is more easily integrated in an enzymatic system such as the DNA toolbox.

We then move on to Computer-Assisted Design. Some DNA computation paradigms have already benefited from the CAD approach, such as VisualDSD, a program developed by Microsoft Research which simulates DNA Strand-Displacement systems [96]. However, the DNA toolbox, while being particularly well suited for computer-assisted design due to its modularity, still lacks such tools. For this reason, we introduced DACCAD, a CAD application for the DNA toolbox. Chapter 4 will present the software in details, as well as a state of the art in CAD tools for molecular programming. Chapter 5 will give an introduction on how to use DACCAD. In Chapter 6, we will see how our model of the DNA toolbox can be used and extended to debug systems.

Then, the next logical step is to automate the process of trial-and-error at the system design level, using an evolutionary strategy to create complex systems. In Chapter 7, we describe BioNEAT, one such strategy, and give a particular example where we developed DNA systems able to "play" the rock-paper-scissors game. Chapter 8 will present the general conclusion to this work, as well as some future avenues of research.

1.7 Glossary

Regroups some important terms in one convenient place.

- Coaxial-stacking: In the situation where two separate but adjacent strands are hybridized to a common third strand (such as the conformation after the action of nickase), coaxial-stacking represents the (often) stabilizing effect of the two stands on each other. It can be seen as both strands interlocking into the usual double-helix, making the whole structure more stable than expected.

- Denaturation: In the case of DNA, the action for a double-stranded sequence to detach. If no sequence remains double-stranded in a given DNA molecule, the two strands are free to move apart.

- Displacement: The action of replacing a strand of a double-stranded DNA molecule by another. This can be done by branch migration in the case of DNA-based strand-displacement, or by the action of the polymerase enzyme if it has strand displacement activity.

- DNA toolbox: A set of three modules (activation, autocatalysis, inhibition) introduced by Montagne *et al.* to mimic gene regulatory networks.

- DNA computing: The general programming principle of using DNA molecules (and possibly enzymes) to encode computation.

- dNTP: A single unit of DNA. Used by the polymerase to build new DNA chains.

- Double-stranded: Two strands attached together to form the famous double-helix structure.
- Evolutionary Algorithm (EA): an optimization strategy based on the concept of evolution. A large number of possible solutions, named *individuals* is generated. Solutions are then ranked based on how good they are. Good solutions are then allowed to "reproduce" while bad solutions die out, generating the next set of solutions.
- Exonuclease: An enzyme that "eats" single-stranded DNA into inactive pieces. It is possible to modify the DNA backbone to prevent its action.
- Hairpin: Loop structure where a DNA strand is hybridized to itself, which gives it a hairpin shape, hence the name.
- Hybridization: The act, for two single-stranded complementary sequences, to attach together.
- Hydrolysis: The act of destroying DNA into monomers (singletons). See exonuclease.
- Individual: In an Evolutionary Algorithm, a potential solution to the problem at hand. The name "individual" comes from the analogy between the algorithm and the evolution of species.
- Inhibitor: In the DNA toolbox, a medium-sized DNA strand that will attach to a target template, inactivating it.
- *In-silico*: Simulated experiment.
- *In-vitro*: Experiment conducted in a test tube.
- *In-vivo*: Experiment conducted in a living environment (cell, bacteria, embryo, tissues).
- Mismatch: An incompatible nucleotide match in two sequences that would be otherwise complementary. Note that a single mismatch does not, in general, prevent the hybridization as long as the matched section is long enough.
- Molecular programming: Using chemical reactions as a paradigm for computation. Chemical species act as the intermediary or final results of a given program.
- Nicking enzyme: An enzyme that recognizes a specific double-stranded DNA sequence, attach to it, and cut one of the backbones at an enzyme-dependent position.

- Nucleotide: In DNA, one of A (Adenine), T (Thymine), G (Guanine) and C (Cytosine), or possibly an artificial equivalent.
- Polymerase: An enzyme able to attach to the end $3'$ end of a hybridized strand and extend it by matching the opposite strand.
- Primer: A short strand used to initiate the action of the polymerase.
- (DNA) strand: A backbone holding nucleotides. Two hybridized strands are required to form the DNA double-helix.
- Template: A long DNA strand defining the sequence that directs the replication by a polymerase enzyme. In the DNA toolbox, templates are protected DNA strands with two domains that encode the reaction system.
- Toehold: A short single-stranded DNA domain that allows a target DNA molecule to dock, possibly to initiate a strand-displacement.
- Turing machine: An abstract machine defined to be a model of what a computation is.
- Turing universal: The property of a computation paradigm to be able to perform any calculation an abstract Turing machine could do. Proving this property is usually done by exhibiting how to program a Turing machine, or any other Turing universal mechanism, in said paradigm.

Chapter 2

Modeling the DNA toolbox, a lightweight implementation for DNA computing systems

This Chapter presents the DNA (Dynamic Networks Assembly) toolbox, central to the work presented in this thesis. The first section gives a general view of its working, which should be enough to understand the remaining of this thesis. Next comes a complete description of the model we are using in the subsequent Chapters. This model is a formalization and extension of that of Padirac *et al.* [11]. This Chapter also presents a comparison between experiment and model as well as some interesting results based on the analytical resolution of the equations of an autocatalytic module at equilibrium.

Our model and the comparison with experimental results have been submitted as part of reference [97].

2.1 General

The DNA (Dynamic Networks Assembly) toolbox was introduced by Montagne *et al.* to help rationally design DNA-based molecular programs, and demonstrated its effectiveness through multiple systems such as the Oligator [67]. It has the advantage of using very simple modules (activation and inhibition) which can be combined to form, in theory, arbitrarily complex systems. Moreover, the DNA toolbox is only based on DNA interactions as well as three specific enzymes, removing the need for intermediaries, such as RNA and protein expression. In the toolbox, two kind of DNA strands have to be distinguished: short DNA strands are used as *signal*, and longer strands are used as *templates* to generate new signal strands (Figure 2.2). *Activation* is done by a short strand being extended by a DNA polymerase enzyme along a compatible template (see Figure 1.4, d.), after which both the signal and output strand are cleaved apart by nickase (see Figure 1.4, e.). It should be noted that the same nicking enzyme is used for all templates in the standard version of the DNA toolbox, which increases the modularity of the system at the price of reducing the sequence design space. This reduction comes from the mandatory introduction of the nicking recognition site, which makes up more than a third of the total DNA sequence of signal strands in the current implementation of the DNA toolbox. Allowing multiple nicking enzymes would ease this problem, as well as allowing finer control on reaction rates, but experimental settings would be much more complex due to the necessity to find buffers fitting all enzymes. Specifically, different enzymes usually work best in different conditions, so the more enzymes, the harder it gets to find conditions appropriate to all of them. Additionally, Padirac *et al.* argue that the sequence design space is still large enough for most reasonable applications [43][1].

After being cut by the nickase, both signal and output are too short to form stable duplexes at the working temperature and are eventually *denatured*, releasing the strands. Their stability is however higher when they are both attached to the template, forming a nicked full duplex, due to a phenomenon known as *coaxial stacking* [98]. This phenomenon is due to the two nucleotides at the nicking site locking each other in the common double-helix shape.

Autocatalysis is a specific case of activation in which both domains of the template are the same, leading to a logistic increase of the signal strands, that is an exponential growth with a saturation factor (see, for instance, Murray [99]) due, here, to the concentration of template and enzymes.

Inhibition is done by slightly longer strands that are complementary to most of a targeted template, but do not trigger polymerisation, thus inactivating the template. Inhibitor-template duplexes (that is, molecules made up of an inhibitor and its target template attached together, see Figure 2.2, inhibition, right), albeit stronger than signal-template duplexes, are not stable either, which guarantees that inhibited templates are eventually freed. Moreover, inhibitors are leaving a short toehold both on the input and output domains (respectively the green and orange domains in Figure 2.2, inhibition) of the template. This allows input or output signal to remove the inhibition at a slow rate through strand displacement (see previous Chapter).

[1]This brings to mind the famous computer-related (mis)quote “640Ko of memory ought to be enough for everybody”. One can hope that, similarly, ways around the current limitations will be found before they start being a problem.

	Electronics	DNA toolbox
Memory, variables	Registers/RAM	Signal strands
Software	Assembly code	Templates
Hardware	Mother board	Enzymes
Power	Electricity	dNTP (DNA monomers)

Table 2.1: Equivalence electornics/DNA toolbox.

Both signal and inhibition strands are degraded over time by the action of an exonuclease enzyme to keep the system out of equilibrium. Templates are chemically protected against such degradation and as such are stable over time. For this reason, it is easy to make an analogy between computers and the DNA toolbox (Table 2.1): enzymes are the hardware doing operations following the software (templates) updating the value of variables (signal and inhibition strands). To complete the analogy, the reaction buffer, which contains the energy and elements necessary for the enzymes to act, would be the electrical supply, either a battery (closed systems) or a power outlet (open reactors).

In Padirac *et al.*'s implementation [11], on which the default parameters of this thesis are based, signal strands are 11-mers (11 nucleotide-long polymers), while inhibition is done by 15-mers. Inhibitors strands are made of three parts: the first seven nucleotides are complementary to the end of the input strand, the next six are complementary to the beginning of the output strand and the last two are mismatched with the target template to prevent elongation by the polymerase. Note that it could be possible to make the inhibitors occupy the target template in different locations, as long as the nicking recognition and cutting sites are not produced at the same time. This inhibition strategy also makes it hard to have inhibition of templates generating inhibitors, since the output of such templates, being inhibitors themselves, will have a long toehold to displace their inhibitor. Using specific (longer) inhibitors to hide the output domain completely was not deemed a good solution, since those "super" inhibitors, in turn, will require even longer strands, and so on. Additionally, the longer sequence makes those "super" inhibitor too stable, releasing the template too slowly. Instead, it is possible to use an intermediate activation of inhibitor (A generates B which generates the inhibitor) and inhibit the first step (A generates B).

Any system using this paradigm can be easily represented as a graph: signal and inhibition strands are the nodes while templates are represented by an arrow going from their signal to

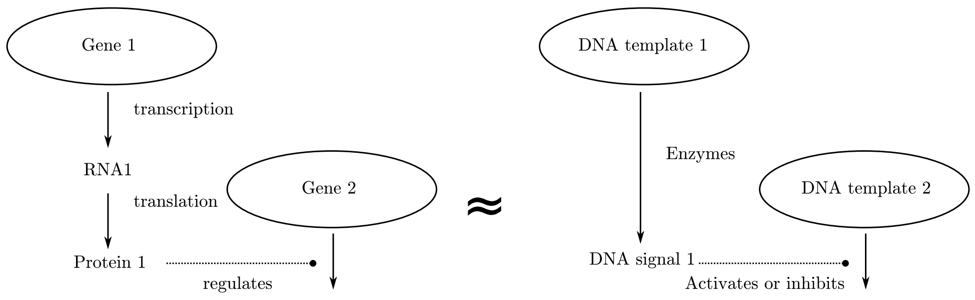


Figure 2.1: Similarity between gene regulation and the DNA toolbox. The DNA toolbox keeps the same structure while only relying on DNA to encode both the structure of the network through templates and the signaling.

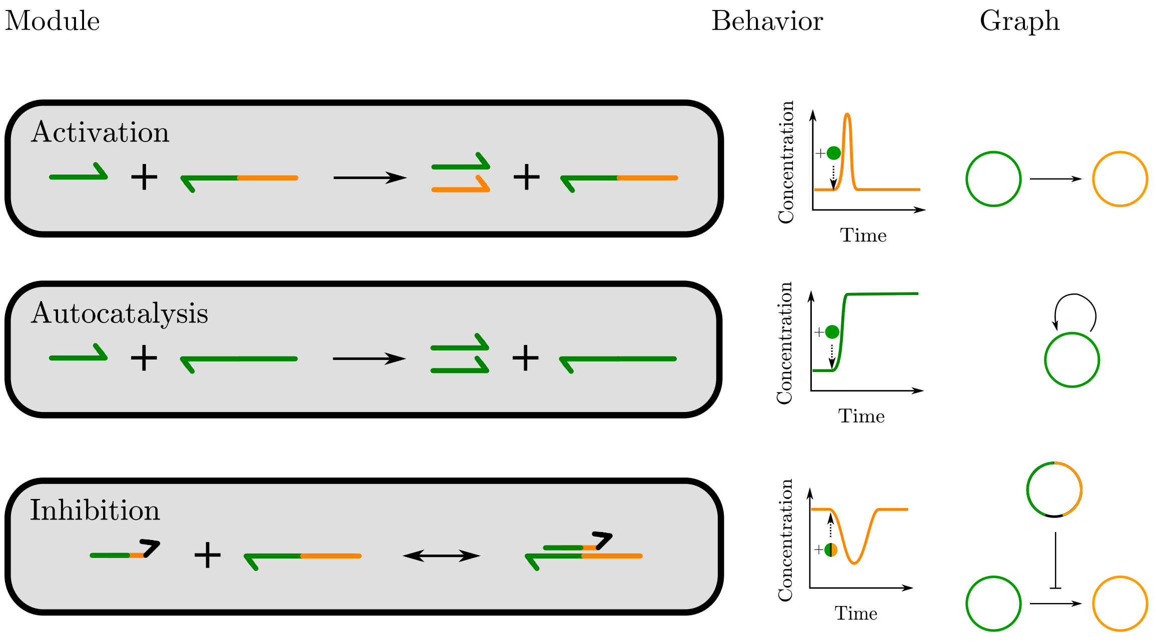


Figure 2.2: Modules of the DNA toolbox: activation, inhibition and the special case of autocatalysis. Behavior shows the impact of injecting some green signal strands in the activation modules. In the case of the inhibition module, we suppose that the inhibited template is at a steady-state, continously generating orange strands. Inhibitor strands are then injected, reducing the output of orange strands. The system goes back to the steady-state when all inhibitors have been consumed by the exonuclease. Graph shows the graphical representation of those modules in the rest of the thesis. Signal and inhibitor strands are represented by vertices and templates by arrows. Inhibition is represented by a bar-headed arrow.

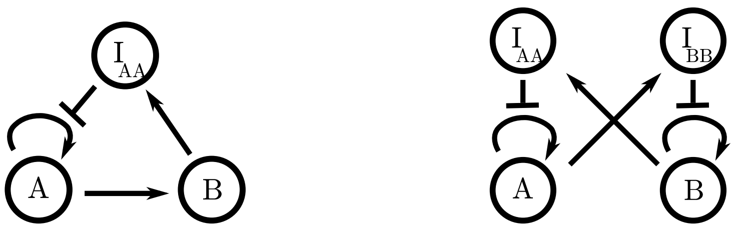


Figure 2.3: Example of graph representation of the DNA toolbox: an oscillator dubbed the Oligator [67] and Padirac *et al.*'s bistable system [11].

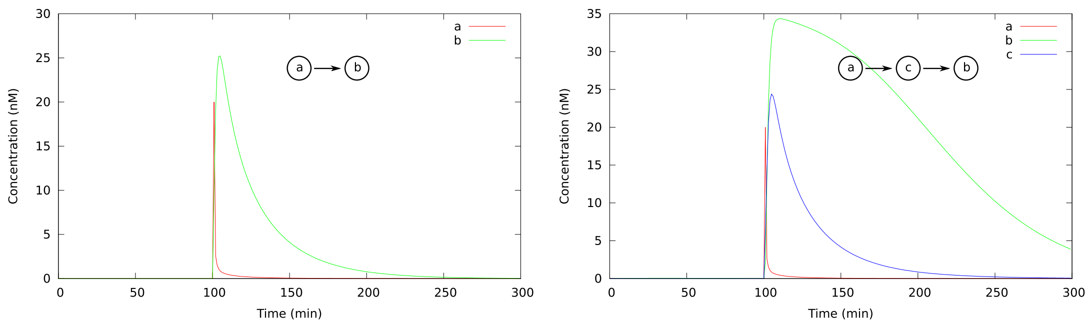


Figure 2.4: Impact of indirect activation of a given sequence.

their output. A second kind of arrow ("bar-headed" arrow) is also used to represent which template is inhibited by a given inhibition sequence. Simple examples of such representation are shown in Figure 2.3. Conversely, any such graph can be directly converted to a DNA toolbox implementation.

However, despite the simplicity of the DNA toolbox paradigm, many effects are very hard to take into account for a human designer. For instance, replacing a direct activation (A promotes B) by an indirect one (A promotes C which promotes B) will result in more than just some latency in the activation: there will also be a latency in all responses of the sequence B, including its overall degradation rate (Figure 2.4). Additionally, enzymes may get saturated, which would change the reaction rates of other parts of the system in ways difficult to apprehend for the human mind [89, 42]. Those problems can be avoided by modifying parameters in the system, or making some additional changes in the structure, operations that can be easily carried out with CADtoolbox (see Chapter 4) or evolved (see Chapter 7). Finally, it should be noted that the DNA toolbox has been formally detailed both from the theoretical [100] and experimental point of view [101].

2.2 Mathematical modeling

Our model is an extension of the one previously developed by Padirac *et al.* [11, 43]. Padirac *et al.* introduced a fairly descriptive and complete model of specific DNA toolbox systems, the bistable circuit and the toggle-switch circuit.

Our contribution is twofold. First, we formalized the kinetic description of DNA toolbox systems. That is, we associated with each module (activation, autocatalysis and inhibition) a set of equations that describes its behavior. This mathematical description allows us to derive the differential equations of arbitrary combinations of modules. Our second contribution is the addition of new details and parameters, such as coaxial-stacking. Those parameters allow us to get a more predictive model without adding much complexity. This also includes a way to express the coupling between the elements in a system, based on enzymatic saturation. This saturation is computed by summation over all possible enzymatic substrates in the system, using the formula proposed by Rondelez [89]. This enzymatic expression is at the same time more realistic than first order kinetics, while remaining easy to estimate. An alternative possibility would be to consider the actual concentration of enzymes and simulate explicitly the intermediate interactions with the different substrates. For instance, one could consider updating the concentration of free polymerase based on how fast it attaches to duplexed DNA as well as how fast it is freed. However, those reaction speeds are hard to estimate, making such model hard to use.

Instead, we try to keep our model of the DNA toolbox simple, yet descriptive enough. For this purpose, we consider reactions at the domain level (highest level of abstraction on Figure 1.3). Such level abstracts actual DNA sequences and instead considers only meaningful interactions, from the designer point of view. In particular, in the DNA toolbox, signal and inhibition strands are composed of only one domain, so we consider they are either completely free or completely attached to their target. Similarly, templates are composed of two domains, since they have both an activation and an output site. This approach has proved itself over time in DSD (DNA Strand Displacement) systems [96, 102, 41], and can be extended to the DNA toolbox if we consider that enzyme activity is carried out as a single step. Particularly, this means that no intermediate product of an enzymatic reaction, such as a partially extended DNA strand, can interact with the system. This also means that the evolution of the state of a module over time can be derived only from the current concentrations of the relevant sequences and its current state (Figure 2.5).

With those restrictions, the set of possible reactions taking place is large, but straightforward: both signal and output sequences can attach to or detach from the template and an inhibition sequence can displace them if there is a toehold available (i. e. if they are not both attached to the template), the polymerase enzyme will extend the signal strand, displacing

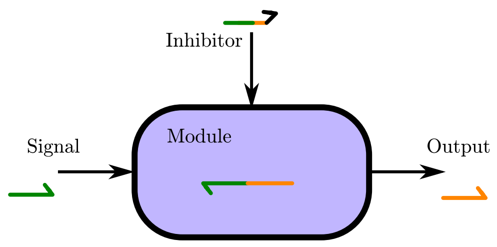


Figure 2.5: An activation module from Figure 2.2 as a black box, with its three external component: the signal, inhibition and output strands. Based only on the current concentrations of those three elements and the module current state, it is possible to compute the module's impact on said elements, as well as the derivative of its state.

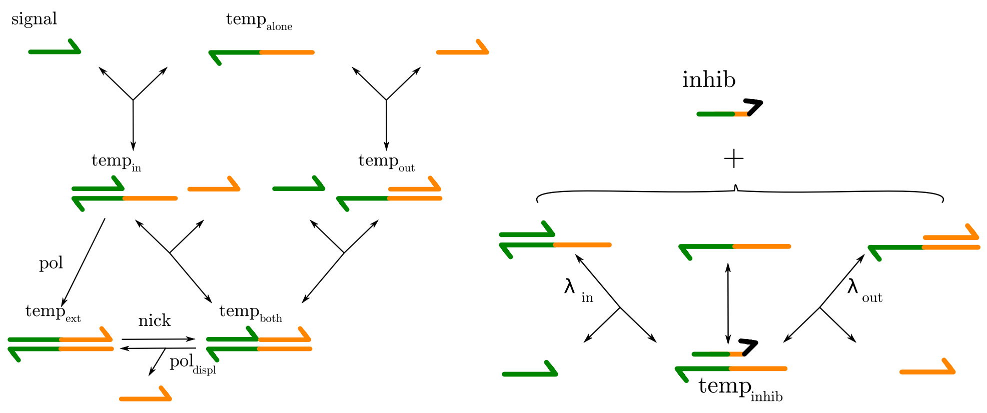


Figure 2.6: All the possible reactions between a template and its various inputs. The respective concentrations of the different configurations of template represent the state of a module *temp*, noted $temp_{alone}$ (template alone), $temp_{in}$ (signal strand attached), $temp_{out}$ (output strand attached), $temp_{both}$ (both signal and output strands attached), $temp_{ext}$ (template completely double-stranded) and $temp_{inhib}$ (inhibited template). Left: working of an activation template without inhibition. Right: reactions related to inhibition. λ_{in} and λ_{out} represent the fact that signal and output strands, respectively, invade an inhibited template at a slower rate since they have a very short toehold.

the output if it was still attached and so on. The set of possible reactions for a non-inhibited template as well as inhibition-related reactions are shown in Figure 2.6. Other DNA strands in the system are supposed to perfectly avoid interactions. Considering that they have at least a few shared nucleotides, this supposition might be false in large systems, leading to additional inhibitions or first-order leaks (signal strands producing output on the wrong template). Those are considered in Chapter 6, where we try to "debug" experimental systems.

The derivative of the concentration of a signal strand s can then be deduced from its interactions with all relevant templates present in a given system as follows:

$$\frac{d[s]}{dt}(t) = \sum_{temp \in I} \phi^{in}_{temp}(t) + \sum_{temp \in O} \phi^{out}_{temp}(t) - exo_s(t) \cdot [s](t) \tag{2.1}$$

where I is the set of all modules accepting s as activator, O the set of all modules generating

s, $\phi^{in}_{temp}(t)$ the flow of s as input of a template *temp*, $\phi^{out}_{temp}(t)$ its contribution as the output of a template *temp* and $exo_s(t)$ the activity of the exonuclease enzyme respectively to s at time t, which is a constant depending only on s if enzymatic saturation is not taken into account. Furthermore, based on the measures of Montagne *et al.* [67], we assume that exo_s mostly depends on the length of s, which means it can only take two values: one for the signal strands and one for the inhibition strands. Except for this strand-specific variation, enzymatic activity is considered first order in the basic model, meaning in particular that exo_s does not change over time. Also note that a given module can be both in I and O in the case of autocatalysis. $\phi^{in}_{temp}(t)$ and $\phi^{out}_{temp}(t)$ can be directly derived from the current state of the module *temp*:

$$\begin{aligned} \phi^{in}_{temp}(t) &= k_{duplex}K_s([temp_{in}](t) + stack_{temp}[temp_{both}](t)) \\ &\quad + k_{duplex}[inhib_{temp}](t)[temp_{in}](t) \\ &\quad - k_{duplex}[s](t)([temp_{alone}](t) + [temp_{out}](t) + \lambda_{in}[temp_{inhib}]) \end{aligned} \tag{2.2}$$

All bracketed terms are concentrations. k_{duplex} is the kinetic constant of association, that is, the constant representing how fast two complementary DNA strands will form a double helix. It is considered sequence independent, and its default value is based on Zhang's measures [61]. K_s is the ratio of the kinetic constant of the reverse reaction over k_{duplex} and can easily be obtained by experiment or nearest-neighbour model. $stack_{temp}$ is the dimensionless factor representing the increase in stability due to coaxial stacking between the input and output when they are both attached to the template. λ_{in}, as well as λ_{out} in the following equations, represent the dimensionless slowdown due to a signal strand invading an inhibited template. The two different values are based on the size of the toehold left by the inhibitor on the relevant side of the template. The first term represents the signal strands restored by detaching from a template. The second term represents those restored by being displaced by an invading inhibitor. The last term comes from the reaction opposite to that of the first term and represents signal strands captured by templates.

$$\begin{aligned} \phi^{out}_{temp}(t) &= k_{duplex}K_s\left([temp_{out}](t) + stack_{temp}[temp_{both}](t)\right) \\ &\quad + k_{duplex}[inhib_{temp}](t)[temp_{out}](t) \\ &\quad - k_{duplex}[s](t)\left([temp_{alone}](t) + [temp_{in}](t) + \lambda_{out}[temp_{inhib}](t)\right) \\ &\quad + pol_{displ}(t)[temp_{both}](t) \end{aligned} \tag{2.3}$$

The first three terms mirrors those of the previous equation. The additional term represents output strands released by the displacement activity of the polymerase enzyme when it extends a nicked duplex. pol_{displ} represents the polymerase activity when displacing a substrate. When this substrate is long enough (for instance an inhibitor), this activity is slowed down by an additional factor, specific to the length of the substrate. If not, the increased affinity for the polymerase is considered to be offset by the slowdown, so that $pol_{displ} \simeq pol$. Note that the polymerase activity *pol* only appears in the equations relative to the internal update of templates (see below).

The derivatives for inhibition strands are similar, but the input term is replaced by an inhibition term:

$$\frac{d[i]}{dt}(t) = \phi^{inhib}(t) + \sum_{temp \in O} \phi^{out}_{temp}(t) - exo_i(t)[i](t) \tag{2.4}$$

Note that there is only one flux for the inhibition term since a given inhibition strand targets a specific module *temp*:

$$\begin{aligned} \phi^{inhib}(t) &= \alpha k_{duplex} K_i [temp_{inhib}](t) \\ &\quad - k_{duplex}[i](t)\left([temp_{alone}](t) + [temp_{in}](t) + [temp_{out}](t)\right) \\ &\quad + k_{duplex}[temp_{inhib}](t)\left(\lambda_{in}[s_{in}](t) + \lambda_{out}[s_{out}](t)\right) \end{aligned} \tag{2.5}$$

Where α represents the fact that an inhibition strand is less stable on its target than on the template that created it due to sequence mismatch (see figure 2.2).

Finally, the equations for the update of a module *temp* are as follows:

$$\begin{aligned} \frac{d[temp_{alone}]}{dt}(t) &= k_{duplex}\left(K_{in}[temp_{in}](t) + K_{out}[temp_{out}](t) + \alpha K_{inhib}[temp_{inhib}](t)\right) \\ &\quad - k_{duplex}[temp_{alone}](t)\left([s_{in}](t) + [s_{out}](t) + [s_{inhib}](t)\right) \end{aligned}$$

$$\begin{aligned} \frac{d[temp_{in}]}{dt}(t) &= k_{duplex}[s_{in}](t)\left([temp_{alone}](t) + \tau[temp_{inhib}](t)\right) + k_{duplex}K_{out}[temp_{both}] \\ &\quad - k_{duplex}[temp_{in}](t)\left(K_{in} + [s_{out}](t) + [s_{inhib}](t)\right) - pol(t)[temp_{in}](t) \end{aligned}$$

$$\begin{aligned} \frac{d[temp_{out}]}{dt}(t) &= k_{duplex}[s_{out}](t)\left([temp_{alone}](t) + \tau[temp_{inhib}](t)\right) + k_{duplex}K_{in}[temp_{both}] \\ &\quad - k_{duplex}[temp_{out}](t)\left(K_{out} + [s_{in}](t) + [s_{inhib}](t)\right) \end{aligned}$$

$$\begin{aligned} \frac{d[temp_{both}]}{dt}(t) &= k_{duplex}\left([s_{in}](t)[temp_{out}](t) + [s_{out}](t)[temp_{in}](t)\right) + nick(t)[temp_{ext}](t) \\ &\quad - k_{duplex}[temp_{both}](t)\left(K_{in} + K_{out}\right) - pol_{displ}(t)[temp_{both}](t) \end{aligned}$$

$$\frac{d[temp_{ext}]}{dt}(t) = pol(t)[temp_{in}](t) + pol_{displ}(t)[temp_{both}](t) - nick(t)[temp_{ext}](t)$$

$$\begin{aligned}\frac{d[temp_{inhib}]}{dt}(t) &= k_{duplex}[s_{inhib}](t)\left([temp_{alone}](t) + [temp_{in}](t) + [temp_{out}](t)\right)\\ &\quad -k_{duplex}[temp_{inhib}](t)\left(\alpha K_{inhib} + \tau[s_{in}](t) + \tau[s_{out}](t)\right)\end{aligned}$$

While this basic model does not take into account enzyme saturation, preferring first order activity, this assumption is not realistic for complex systems. Specifically, all modules are sharing the same three enzymes, which is expected to have an impact on their activity. The usual way to model such burden is to use a Michaelis-Menten term to quantify the enzymatic activity. Such a term is further modified to take into account that multiple modules are competing for those resources [89]. The exonuclease term becomes:

$$exo_s(t) = \frac{V_m}{K_m^s\left(1 + \sum_{s' \in seq} \frac{[s'](t)}{K_m^{s'}}\right)} \tag{2.6}$$

where V_m is the maximum theoretical rate and K_m^s the Michaelis constant for the sequence s. Note that in our model V_m is independent of s and K_m^s can only take one of two values, depending on whether s is an inhibition sequence or not. This is based on the experimental observation the value of K_m^s of the exonuclease depends primarily on the length of the substrate s.

The polymerase activity is separated in two terms: *pol* for templates with input alone and pol_{displ} in the case where both input and output are present. We suppose that the polymerase does not interact noticeably with other states of the template.

$$\begin{aligned}pol(t) &= \frac{V_{m,pol}}{K_{m,pol}\left(1 + \sum_{temp} \frac{[temp_{in}](t)}{K_{m,pol}} + \frac{[temp_{both}](t)}{K_{m,displ}}\right)}\\ pol_{displ}(t) &= \frac{V_{m,displ}}{K_{m,displ}\left(1 + \sum_{temp} \frac{[temp_{in}](t)}{K_{m,pol}} + \frac{[temp_{both}](t)}{K_{m,displ}}\right)}\end{aligned} \tag{2.7}$$

Note that $V_{m,displ}$ depends on the length of the output. Based on the experimental results from Padirac *et al.* [11, 43], we further consider that *pol* and pol_{displ} are equal for signal sequences. This means that the displacement slowdown for short outputs compensates the higher affinity of the polymerase for double-stranded substrates.

The nickase term is the simplest. We consider that only fully double-stranded templates can capture this enzyme, yielding:

$$nick(t) = \frac{V_{m,nick}}{K_{m,nick}\left(1 + \sum_{temp} \frac{[temp_{ext}](t)}{K_{m,nick}}\right)} \tag{2.8}$$

We can also note that Padirac [43] showed that this restrinction to $temp_{ext}$ is fairly nicking enzyme dependent, and might be hard to guarantee in the general case.

The complete set of equations describing a simple autocatalyst in given in Appendix A.2, highlighting the actual complexity of writing such systems by hand.

It is also possible to toggle some additional effects to this model, making it more simple or detailed, if necessary: removing enzymatic coupling, in which case saturation terms from substrates other than s are neglected; taking free templates into account in the saturation term of the exonuclease (even though they are not degraded, they can still bind to the exonuclease enzyme and act as competitive inhibitors); forbid signal and output strands to invade inhibited templates; allowing the polymerase to generate output at a very slow rate from templates without primer (phenomenon called zeroth order leak [103]).

Default parameters for simulation were obtained from the literature as follow. Association constants and strand-displacement speeds are based on Zhang *et al.*'s values [61]. Strands dissociation constants and Michaelis-Menten parameters for the enzymes are taken from Padirac *et al.*'s measurements [11]. The value of enzyme activities are also taken from Padirac *et al.*'s experiments, except for the nickase activity, for which the fitted value was taken instead. This choice comes from the high discrepancy between measured and fitted values in those cases, which might indicate a fit strongly dependent on the model. This is due to strong measured variations between batches of enzymes, so fitting those parameters should be the first priority for the user. Note that the user can also play with the enzymatic activity *in vitro* by using different dilutions, depending on the batch. The value for coaxial stacking was trickier, as there is not only a strong sequence dependence [98], but this dependence is not even limited to the nearest-neighbours [104]. To get realistic values, we considered the opening/stacking energy described by Frank-Kamenetskii's group [105], with salt and temperature (42°C) corrections [106] to match the experimental values of the DNA toolbox, giving a range of values between 0.04 (25 times slowdown for a nicking site GC) to 1 (no slowdown for a nicking site TA). The values corresponding to all 16 first-neighbor possibilities are listed in Appendix A.1.

2.3 Correctness of the model

This model can be described as a brute force model: additional details and reactions are included when they become relevant, that is, when the model and experiments are not in good agreement anymore. Since this model was developed to help the implementation of *in vitro* systems, it is necessary to pit it against the reality of experiments. Verifications were thus performed by comparison with the data from Padirac *et al.* [11]. However, comparison is not completely straightforward, as the data are based on observed fluorescence, which can be either obtained by EvaGreen, a molecule that emits light based on the concentration of double-stranded DNA, or by FRET [107], neither of which reflecting directly specific concentrations.

EvaGreen is an intercalating dye, that is, a molecule that insert itself in the double helix of double-stranded DNA and then emits light. The fluorescent light emitted can then be measured to estimate the total amount of double-stranded DNA in the system. However, it is subject to background noise, single-stranded DNA fluorescence and saturation (once all molecules are interacting with DNA, no more increase in DNA concentration can be monitored). In the following, we are making the hypothesis that the concentration of EvaGreen is much larger than the concentration of double-strands in the solution, avoiding any saturation effect. Then, the fluorescence of EvaGreen is linear with the concentration of double-stranded template. FRET (Fluorescence Resonance Energy Transfer), on the other hand, is used in the DNA toolbox by adding a fluorescent molecule at the $3'$ end of a template. Padirac *et al.* showed that when the nucleotides closest to the fluorescent molecule are bonded, there is a noticeable shift in fluorescence [108]. Thus, it is possible to monitor the state of particular template species in real time. More formally, we can introduce the fluorescence measures:

$$\varphi_{eva}(t) \propto \sum_{temp} [temp_{both}](t) + [temp_{ext}](t) + \frac{1}{2}\left([temp_{in}](t) + [temp_{out}](t) + [temp_{inhib}](t)\right)$$

$$\varphi_{fret}(t) \propto [temp_{in}](t) + [temp_{both}](t) + [temp_{ext}](t) \tag{2.9}$$

The factor $\frac{1}{2}$ in $\varphi_{eva}(t)$ comes from the contribution of DNA molecules is based on the number of double-stranded nucleotides, which is half for $temp_{in}$ and $temp_{out}$, and roughly half for $temp_{inhib}$. This measure can be used as additional data when monitoring a system.

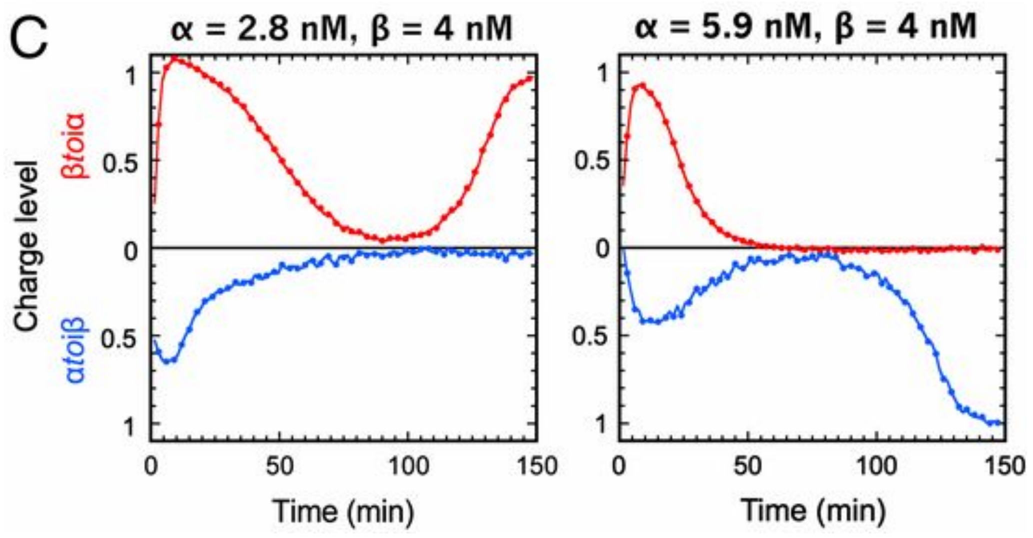
In $\varphi_{fret}(t)$, the expression is only dependent on the templates where a fluorescent molecule is attached. To be correct, this expression should also take $[temp_{inhib}](t)$ into consideration, with a factor corresponding to the nucleotide offset of the inhibitor attached to the template

[108]. However, in all experiments we compare our model to, FRET molecules were attached to templates without inhibitors, so we ignore it for simplicity's sake.

Figure 2.7 shows a comparison between the experimental results of Padirac *et al.* [11] and the simulated behavior of the bistable circuit (see Figure 2.3). We used coaxial-stacking slowdown values based on the DNA sequences given in their work. Additionally, based on Padirac *et al.*'s discussion of possible differences in activity of the polymerase and the experimental values of Qian *et al.* [109], templates relative to one species (β in their article) were given a concentration bonus, due to their expected higher affinity with the polymerase. Simulation results for different bonus are shown alongside Padirac *et al.*'s experimental results (Figure 2.8). While there was good agreement from a qualitative point of view, the stability diagram was slightly shifted (Figure 2.8). Particularly, without template correction, the system is still bistable but the attraction basin of the dominating β state is too small to appear on the Figure. This difference may be due to the inaccuracy of the coaxial-stacking slowdown or of the Michaelis-Menten constants. This is a realistic assumption, since some of the latter come from fitted values based on Padirac *et al.*'s model [11], and as such might not be correct in our model. In both cases, we are using the most realistic values available. For general application, experimental parameters should be measured for a specific experimental setting before this model is used to predict behaviors (such as those in Chapters 4 and 7). We can finally conclude that our prediction is close enough to reality to be used as a guide to design systems, assuming that parameters are carefully fitted to one's experimental setting.

2.4 Fitting parameters

The simplest, and probably best, way to fit parameters to make the model as predictive as possible is to conduct a set of basic calibration experiments. Among the experiments described by Montagne *et al.* [67], the model is particularly adapted to turnover experiment. In a turnover experiment, an autocatalytic module is put in a reaction environment with a low concentration of dNTPs. When all dNTPs are consumed, the polymerase will not be able to sustain the creation of the autocatalytic species, so the system will quickly go to a ground state (no signal species in the solution). Knowing the concentration of dNTP in the solution, it is possible to approximate the first order activity of the polymerase from the time it took to consume all of them. Note that this is only possible if the polymerase is limiting (high template concentration). If not, its first order activity can be obtained from additional experiments. Finally, the ground



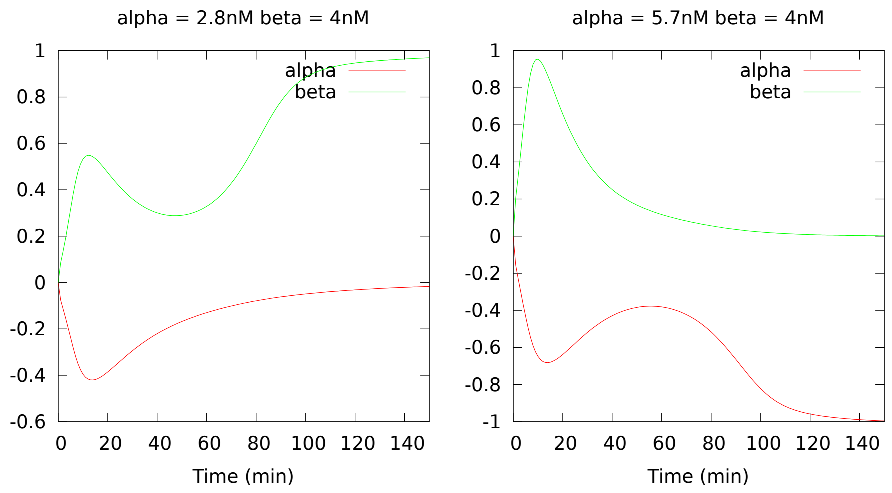


Figure 2.7: Comparison with the experiments conducted by Padirac *et al.* (Figure reproduced from [11] with the consent of the authors). In their system, two signal species, α and β are competing against each other, so that only one can be present at the steady-state. The values used for the simulations are the same as their fitted values [11], and the output was normalized in a similar way. The slight difference might come from different global enzymatic activities or phenomenon not taken into account by the model, such as first order leaks or some sequence-dependent variations in enzymatic affinity.

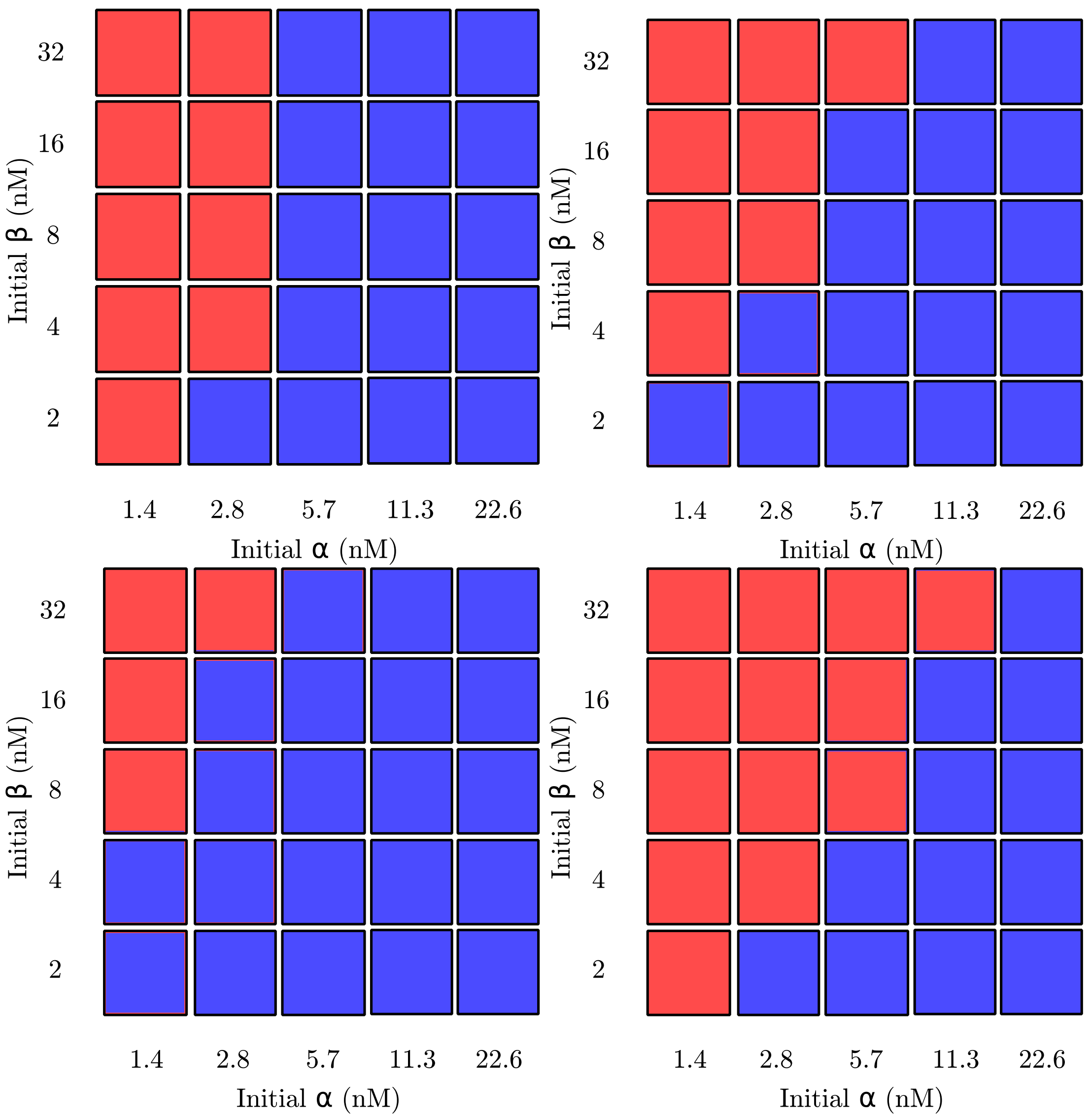


Figure 2.8: Bistability domains of Padirac *et al.*'s experiment [11] (top left) and of simulations with various template concentration bonuses: 15% (bottom left), 20% (top right) and 25% (bottom right). α and β are the same as in the previous Figure. The color of a cell, blue for α and red for β, represents which species wins over the other with those specific initial concentrations. Without bonus, α always win over the considered range, which could be explained by incorrect coaxial stack slowdown and/or polymerase affinity.

state is also important to check the exonuclease activity and get the baseline fluorescence. This baseline has to be substracted before the data can be analyzed in any quantitative way. Additionally, in the case of a single autocatalyst *temp* amplifying a sequence s, it is possible to solve the equations at equilibrium (that is, when all derivatives are equal to zero, at the transient steady-state), in two cases:

In the case of coaxial-stacking slowdown = 1 (no slowdown, possible by choosing carefully the sequence at the nicking site):

$$[temp]_{total} = K_s \cdot \frac{exo_{s,eq}}{pol_{eq}} + exo_{s,eq}\left(\frac{1}{pol_{eq}} + \frac{1}{nick_{eq}}\right)[s]_{eq} \tag{2.10}$$

with $[temp]_{total}$ the total concentration of template and eq indicating values at equilibrium. Note that this formula doesn't take into account zeroth order leaks, as the additional term would have a contribution below what can be observed with a PCR machine.

In the case of coaxial-stacking slowdown = 0 (DNA duplexes are as stable with or without the nick, unrealistic but can be approximated by using G-C nicking sites[2] and having a high concentration of polymerase compared to that of nickase):

$$[temp]_{total} = exo_{s,eq}\left(\frac{1}{pol_{eq}} + \frac{1}{nick_{eq}}\right)[s]_{eq} \tag{2.11}$$

We can note that both equations are very similar, but no simple formula can be obtained in the general case. The only possibility is to rely on numerical resolution to solve the system, using DACCAD for instance. All other parameters being fixed, the effect of coaxial-stacking on the steady-state concentration of the autocatalytic species is shown in Figure 2.9.

Another problem arises from the actual estimation of the species concentration. As mentioned before, it is not possible to measure $[s]_{eq}$ directly. However, it is possible to reach the following result (independent of coaxial-stacking):

$$\begin{aligned} [s]_{eq} &= \frac{nick_{eq}}{exo_{eq}}[temp_{ext}]_{eq} \\ &= \frac{pol_{eq}}{exo_{eq}}\left([temp_{in}]_{eq} + [temp_{both}]_{eq}\right) \end{aligned} \tag{2.12}$$

With $temp_{ext}$ the fully double-stranded template, $temp_{both}$ the double-stranded nicked template and $temp_{in}$ the template with input signal (see Figure 2.6). A proof of those results

[2]A G-C nicking site means that the DNA strand before the nick (in the $5'$ to $3'$ direction) ends with a G, and the DNA strand after the nick starts with a C. This specific configuration gives the highest stability to nicked molecules.

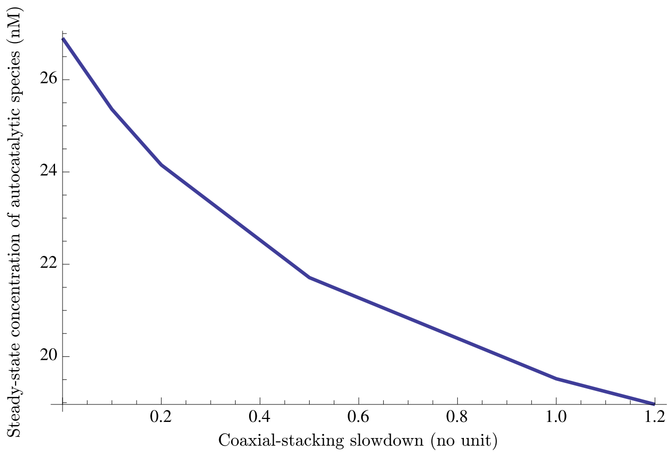


Figure 2.9: Impact of the coaxial-stacking slowdown on the steady-state concentration of an autocatalytic species. The simulation was done for all values presented in Appendix A.1, as well as the hypothetical 0 value (double strand as strong with or without nick). Note that for some configurations the slowdown can be higher than 1, meaning that the resulting molecule is less stable than if there was no stacking. Simulation performed with the default values for enzymatic parameters, 10 nM of template and a dissociation constant of 74 nM.

is given in Appendix A.3. Moreover, those results have also been verified using Mathematica [110], a program for numerical and formal analysis. By using the Equation 2.9, it follows that $\varphi_{fret,eq} \propto exo_{s,eq}\left(\frac{1}{pol_{eq}} + \frac{1}{nick_{eq}}\right)[s]_{eq}$. Note that at the steady state, for enzyme activities similar to that of Padirac *et al.*, most of the template is shared between the $temp_{both}$ and $temp_{ext}$ states. We can then approximate that $\varphi_{eva,eq} \propto [temp_{both}]_{eq} + [temp_{ext}]_{eq}$. A comparison of both values for $\varphi_{eva,eq}$ are shown in Figure 2.10. This approximation yields $\varphi_{eva,eq} \propto 2exo_{s,eq}\left(\frac{1}{pol_{eq}} + \frac{1}{nick_{eq}}\right)[s]_{eq} \propto \varphi_{fret,eq}$. Using equation 2.10 (no slowdown):

$$[temp]_{total} = K_s \cdot \frac{exo_{s,eq}}{pol_{eq}} + cst \cdot \varphi_{fluo,eq} \tag{2.13}$$

where $\varphi_{fluo,eq}$ is the value of any of the two fluorescence that can be measured. Hence, by doing a ramp of concentration for the template, it is possible to fit parameters of the system. The other case (coaxial-stacking slowdown infinite) gives us the multiplicative constant cst and might help giving a quantitative meaning to experimental data.

Note that multiple experiments should be done to average the unavoidable noise. In this case, the redundancy between EvaGreen and the FRET fluorescence gives us an edge, as each run virtual counts as two data sets. One can then fit the relation between the total template concentration and the fluorescence by a linear regression (Figure 2.11), where the baseline is the term $-K_s \cdot \frac{exo_{s,eq}}{pol_{eq}}$. Since the exonuclease and polymerase apparent activities at equilibrium should be independent[3] of s, it is possible to get a fit of the dissociation constant K_s for all

[3]Using parameters from Padirac *et al.*, the model predicts a variation of up to 2.5% of the term $exo_{s,eq}/pol_{eq}$

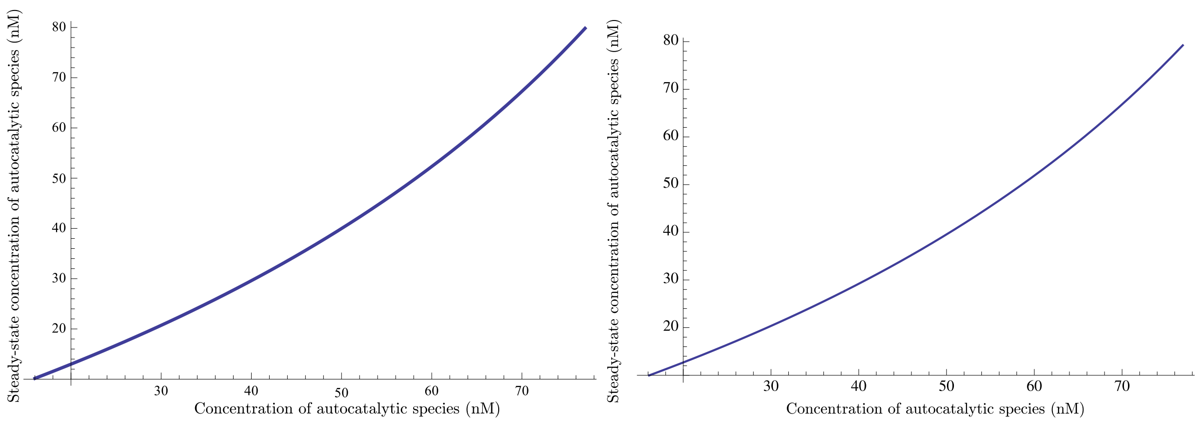


Figure 2.10: Comparison between the theoretical and approximate value of EvaGreen fluorescence. Values where obtained from simulation on an autocatalyst with dissociation constant $K_s = 35$ nM and default enzymatic activity. Left: theoretical value of EvaGreen fluorescence. Right: approximate value. The plots are separated to allow the comparison between the two curves. Over the considered range, the two curves are indistinguishable.

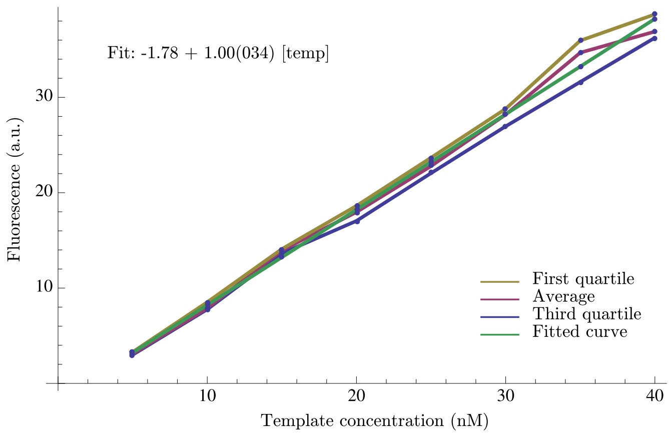


Figure 2.11: Linear regression fit of the average fluorescence of 5 simulated measures for an autocatalyst with a dissociation constant $K_s = 35$ nM. Each simulation was done with 10% noise on the fluorescence signal, representing the inaccuracy of the PCR machine used for monitoring.

signal sequences. On the other hand, if K_s is already known (from nearest-neighbors estimation [111] or otherwise), it is possible to get a first order activity value for the polymerase by measuring the plateau length before all dNTPs are consumed [11] with a high concentration of template (polymerase saturated). Then, it is immediate to get a first order activity value for the exonuclease from the equation $exo_{s,eq} = baseline \cdot \frac{pol_{eq}}{K_s}$. As a proof of concept, we try averaging over five independent (simulated) measures (Figure 2.11). We suppose that the enzymatic activity is known, giving us $K_{s,measured} = 1.78 \times pol_{eq}/exo_{s,eq} \simeq 32$ nM, which is close to the actual value of 35 nM. The difference might also be due to the approximation $pol_{eq}/exo_{s,eq} \simeq constant$.

over a range of concentration of template going from 5nM to 100nM, which should be negligible compared to other measure errors.

Chapter 3

The delay gate

DNA computing has the potential to create powerful devices, but, in the context of well-mixed systems, *sequentiality* of operations is hard to achieve. To enforce such sequentiality, we propose a generic delay gate that can be interfaced with virtually any DNA system. Since it is system-independent, our delay gate can be used as an off-the-shelf library to accelerate the design of increasingly complex systems. Since DNA computing systems, like the DNA toolbox, are Turing-complete, this operation should be possible to implement as a combination of already existing elements. However, such implementation might be relatively heavy in a given paradigm. When designing basic operations, it is important to keep in mind the problem of genericity (whether a building block can be use in a multitude of situation) versus applicability (whether a building block as an impact on the size or speed of a given system). The delay operation having a broad range of potential application, it makes sense to add it as a separate block.

Additionally, we checked the feasibility of our design by testing various *in-vitro* implementations. We also present a theoretical proof of concept of its applicability by using it to complement the DNA toolbox to design new systems. This integration uses a simple Mathematica [110] file describing such extended toolbox systems that can be automatically generated by DACCAD (see next Chapter). The research presented in this chapter has been accepted by the journal Natural Computing (see reference ([112]).

3.1 Introduction

As we mentioned, DNA computing has been used in the recent years to create increasingly complex systems with the ability to perform various tasks, either *in vitro* or *in vivo* [10, 11, 113]. However, the current trend may reach a peak as it gets harder and harder to design more sophisticated devices. In particular, like in a computer, it is important to guarantee the sequentiality of critical operations. The most common illustration of this problem, in computer science, is to imagine two agents and a shared counter. The first agent tries to decrease the counter, while the second wants to increase it. If nothing is done to enforce the sequentiality of those operations (one agent act, then the other), the system may end up in an impossible state: if both agents read the current amount at the same time, then change locally the value before storing it again, both agents will think they acted. However, the state of the counter will only reflect one operation, namely the last which took place. To prevent this, the counter has to be inaccessible to any agent other than the one which is currently trying to modify it.

In test tube DNA computing systems, this problem is omnipresent, as DNA strands used to perform operations are moving freely in the solution. This means they can interact with virtually any other strand at any given time, while doing so could represent an incorrect operation. For instance, we can think of a system where two reactions are in competition, such as the two autocatalytic modules in Padirac *et al.*'s bistable circuit [11]. Even if one is supposed to win over the other, products from the former will still be in the solution as long as the steady state is not reached. Those products may then interact with other modules of the system before the competition is over and produce unexpected behaviors. Leaks coming from such intermediate state of the system or from early product of a given reaction may mislead a comparator, or force a dynamical system such as an oscillator toward a flat steady state. This is also often seen during the initialization of some complex systems where mutual inhibition is not yet fully "operational". The way to solve this sequentiality problem is to use delay or join (arbitrarily long delay) operations. The main difference between those two operations is that *join* enforces sequentiality in the sense of Lamport's logical clock [114], while *delay* only guarantees it in the sense of a wall clock. While the former is stronger from the logical point of view, the latter is enough for a variety of applications where typical reaction times are known. Note that, since there are many factors affecting the reaction rates, such as enzyme saturation or salt concentration in the buffer, an exact timing can hardly be set. This does not limit the usefulness of the gate, as it is enough to ensure the sequentiality, possibly by overshooting the optimal delay.

Since delays are necessary to design non-trivial systems, many workarounds are present in the literature. They are, however, designed with a specific application in mind, which makes them difficult to apply to other systems. Still, the obvious advantage of such approach is that the design of the delay can be specifically optimized for the problem at hand. Nevertheless, we believe that DNA system design could benefit from general all-purpose modules in the same way as off-the-shelf software components work for software development. This belief primed us to develop the delay gate introduced here.

Our gate enforces delays by capturing and sequestering the target sequence (also called output) with a timer strand. Timer strands, as the name implies, represent how long the output will be delayed. Those strands are catalytically degraded over time by our delay gate. Once they have all disappeared, the output signal is completely restored by the gate. Moreover, the gate is designed to have a stronger affinity for single-stranded timer, which means that the output is only released after the delay is over, and that the release speed is not a function of the amount of delay. By playing with the ratio of concentration of output over that of the gate, we can also fix how fast the output will be released, allowing the gate to be adapted to multiple purposes. Note that the target sequence has to be known at the time of design. This does not, however, diminish the usefulness of the gate unless we are working with a system generating random sequences [1]. There are mainly two ways to delay the result of an operation with multiple possible outputs: either sequester a sequence required by the targeted operation, such as its fuel strand, or capture all possible outputs by using as many timer strands. All those timers can then be synchronized by using the same delay gate, which guarantees the same delay regardless of the output.

We present both simulations and experimental results of an actual implementation of the delay gate. We then demonstrate how the gate can be used in collaboration with other systems, such as Montagne *et al.*'s DNA toolbox [67], as well as detailing some original uses that can be made of it.

3.2 Related work

One way to implement delays was proposed by Qian and Winfree [10, 41] (see Chapter 1). In their seesaw gate, they employ a threshold module to prevent early comers from rushing ahead, thus eliminating leaks. The idea of this module is to capture and waste a fixed quantity of input before it had any chance to interact with the rest of the seesaw gate's components. If the threshold is passed, the signal is restored through the use of fuel that regenerates the input until all the output has been released, which allows the seesaw gate to be cascadable. However, in this design the output is released over a long amount of time, which means that long reaction cascades are very time consuming. Additionally, the threshold module is wasted when used,

[1]Such systems might be dangerous, anyway, as they tend to generate parasites [115]

with no mechanism to reliably regenerate it. This means that a given seesaw gate cannot be used twice. In particular, loops are not possible with this design, so that complex computations require a huge quantity of gates.

Another approach was presented by Condon *et al.* [7, 116]. In their case, more than leaks, the problem comes from ensuring that operations are executed in the right order. This problem is well-known in parallel computer programming, where more than one program can access to and modify a shared resource, such as memory. In their implementation of a gray code counter, the state of the counter can be seen as the memory and the various gates necessary to increase it as the concurrent programs. More specifically, the problem was that more than one step was necessary to perform a valid operation on the counter, while other gates could interact with it before all steps were done. They thus borrowed the notion of mutex, a single object used to make complex operations atomic (that is, happening in an uninterruptable manner): only the gate "holding" the mutex could perform operations on the current state. The mutex itself was implemented as a DNA strand necessary to open the toehold of the gates, so that only the gate to which it was attached was active. The immediate drawback of this approach is that the mutex strand has to be unique, *i.e.* a single occurrence of this particular molecule should be present in the system. Even in the case where this condition is achieved, it would cause very low reaction rates, since the mutex has to be compatible with every (non-wasted) gate, while only a limited amount of them are valid to perform the next computation step.

Montagne *et al.*'s oscillator is based on an autocatalitic module with a negative feedback (Figure 2.3, left). They found that having a direct negative feedback only created damped oscillations, with the concentrations of all species in the solution reaching a stable steady state and that a delay was thus necessary [117]. They solved this problem by implementing a longer reaction path for the negative feedback (A gives B gives inhibitor, instead of A gives inhibitor), introducing an implicit delay. See Section 3.6 for additional details. The delay gate, although heavier to use than the simple transduction scheme they use, is more generic.

From a functional point of view, the delay gate is related to the whiplash PCR [69]: a primer is extended by polymerase until a stopping point, before being freed by strand displacement. In the whiplash PCR successive releases and extensions are used to perform computation through the modification of the "whip" part (the gate in our case) of the design. We however use it to release and waste the extended primer (our timer strand) without any modification being made to the gate: we simply use the whiplash mechanism to ensure the catalytic nature of the gate.

3.3 Working principle

The main principle is to use a specific kind of DNA strands, called timer strands, to capture the output to delay. Timer strands are in excess of the output and form stable hybridization duplexes with them, so that the effective concentration of free output in the solution is close to zero. Timer strands are then catalytically degraded by the delay gate. Single-stranded timer strands are consumed first, introducing the delay. Then the timer strands with output are consumed, freeing the latter.

The gate works in four steps, as shown in Figure 3.1, left. (1) Either a single-stranded timer strand, or a duplex output-timer attaches to the gate. By using n nucleotides at the $5'$ end of the output to cover the toehold of the timer-gate duplex, we enforce that single-stranded timer will be consumed first, as the size of the toehold changes the duplex attachment kinetic parameter[61]. (2) DNA polymerase with strand displacement activity (such as Klenow large fragment or Bst large fragment) extends the timer until the stopper, which can be anything that would not be recognized by the enzyme. (3) The displaced end of the gate comes back, releasing the head of the timer to which the fuel attaches itself. (4) The fuel strand is extended by the polymerase, wasting the timer and releasing the gate; if the timer was duplexed, the output is released by this operation as well.

The use of fuel may be inconvenient in some cases, such as when the delay gate should be reused multiple times in a closed system. For this reason, we also designed a fuel-free version of the gate (Figure 3.1, right). In this version, the extended sequence of the timer is partially self-complementary, allowing it to be its own primer for the last step. It should be noted that this version of the gate can also form an additional hairpin at the position covered by the extended timer. However, at least half of the sequence of this hairpin is covered most of the time, either by the $5'$ end of the gate (as in steps (1), (3) and (4)) or by the timer strand (steps (2) and (3)). Moreover, even in the eventuality of the formation of this hairpin, it doesn't prevent the attachment of the timer strand, whose extension will then force it open.

The main challenge of the delay gate is to ensure that the output is released at the right time. This is guaranteed by three different mechanisms. First, since the output covers part of the timer toehold, hybridization of the duplex with the gate is less stable than that of the single-stranded timer. Thus, the hybridization of single-stranded timer with the gate is kinetically favored. Second, it also means that if a duplex is hybridized to the gate, there is still a small toehold for a single-stranded timer to displace the duplex (Figure 3.2). Finally, as a fail-safe,

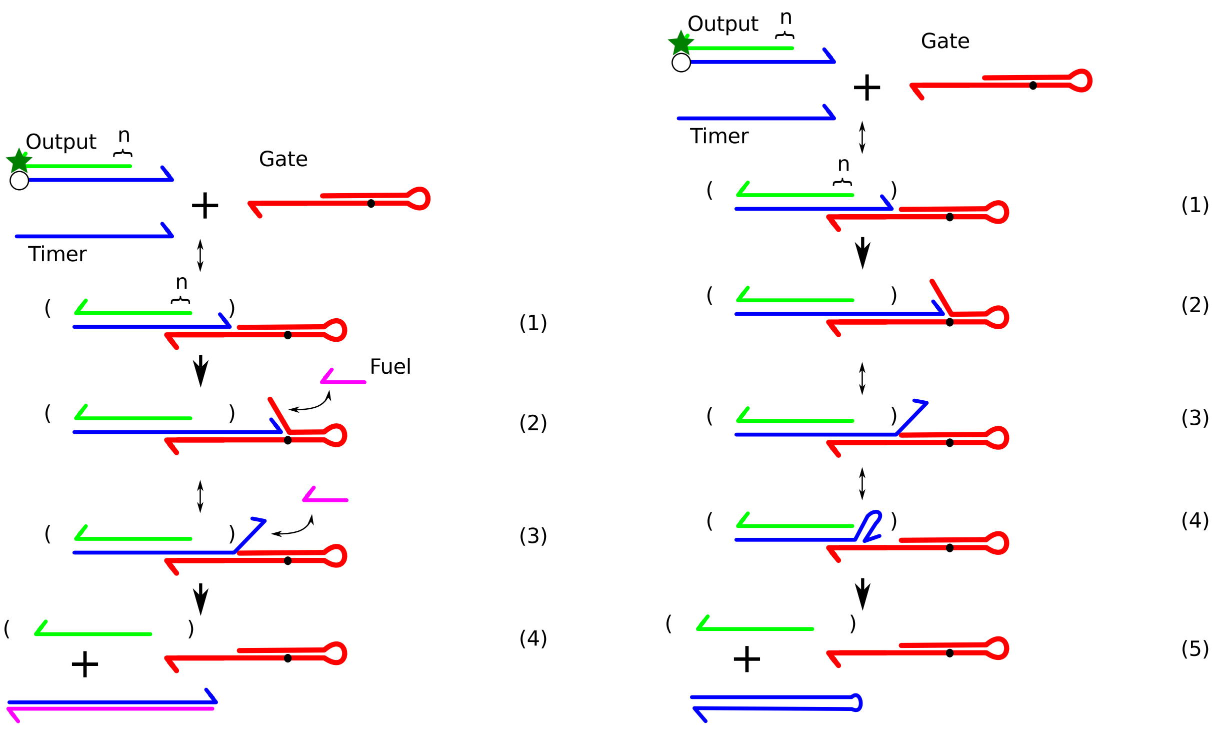


Figure 3.1: Schematic working of the delay gate. Parentheses represent the fact that the output is optional in the configuration, as the working of the gate is the same. Timer without output are consumed first. Left: fuel-driven version; right: fuel-less version. Arrows represent single DNA strands, from their 5′ end to their 3′ end. The black dots on the gate represent polymerase stoppers, for instance missing dNTPs or a polyethylene glycol spacer. n represents the length in bases of the sequence that is common both to the output and the gate. A modification on the 3′ end of the gate prevents it from being extended by using the timer as a template.

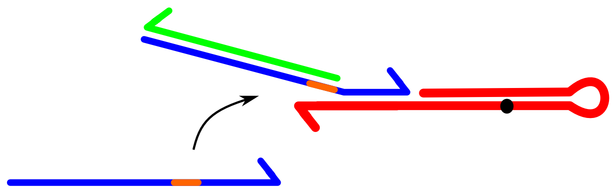


Figure 3.2: Toehold (orange) for the displacement of an output-timer duplex by a single-stranded timer. Note that if enough bases of the output strand are free, the single-stranded timer can capture it instead of invading.

the output can be captured by another single-stranded timer if still present in the solution (i.e. the release was too early). This last point is similar to the threshold gate of Qian and Winfree[10, 41], where the threshold mechanism sequesters part of the input. In the delay gate, however, the totality of the input signal will be eventually released.

Some applications may require to disable and re-enable the gate when necessary. This can be done by using a strand opening the hairpin of the gate, leaving only a short toehold for its complementary to free the gate (Figure 3.3). Note that opening the hairpin may actually requires the gate to be in step (2), which means it is currently consuming timer. This does not limit the effectiveness of this technique since, if there is no timer, disabling the gate has no purpose. One possible issue with the presented disabling mechanism is that the disabling strand might attach to the $3'$-end of an extended timer strand in step (3), causing the later one to be extended again by polymerase (Figure 3.4). There are, however, multiple workarounds to this problem. The simplest one would be to add some mismatch in the sequence of the disabling strand, making it unlikely for the polymerase to extend the timer strand. If the disabling strand is not generated dynamically during the computation, another possibility is to design it with a stopper similar to that of the delay gate, preventing any extension. Finally, the complementary sequence of the disabling strand can be moved past the stopper of the delay gate, with a toehold present in the gate's loop instead that on the whiplash area, removing any interaction with the timer strand.

3.4 Model and simulation

We take into account all the reactions presented in the previous section (see Figure 3.1). All those reactions are considered atomic, that is, indivisible. For instance, the polymerase does not stop until it reaches the end of a strand or a stopper, complementary strands cannot be only partially attached to each other, and in the case of competition, we only consider the states where one or the other possible strands is completely attached, leaving the other completely free.

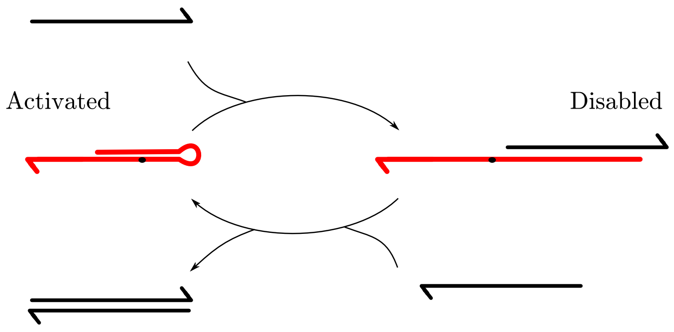


Figure 3.3: Activation and deactivation of the delay gate. Note that the gate is shown in the configuration of the first step of Figure 1, but that the opening of the hairpin can only occur in step (2).

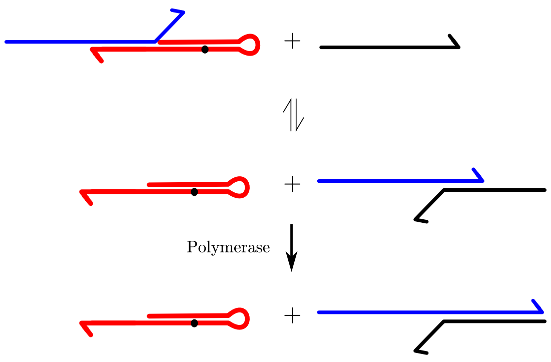


Figure 3.4: Incorrect interaction between a disabling strand and a timer strand. This comes from the fact that the $3'$ end of the extended timer strand and the whiplash section of the delay gate have the same DNA sequence. Note that the disabling strand still has a toehold to accept an activation strand.

We also do not take into account possible crosstalks other than that of the fuel with the freed end of the delay gate (see step (2) in Figure 3.1) as they can be avoided by careful sequence design. Polymerase saturation is also considered negligible and we model its activity using first order kinetics [118, 42]. For a full integration with the model of the DNA toolbox presented in the previous Chapter, additional experiment determining the actual Michaelis-Menten parameters of the gate would be necessary, but the saturation-free model should be enough for a variety of application.

We then extract from the full set of reactions (see Appendix A.4) the system of differential equations, assuming mass action kinetics. We consider two sets of kinetic parameters for the numerical solving of this system. The first one, dubbed generic, was defined to be qualitative, and only reflects the magnitude of differences between reaction rates (with possible values being *very slow*, *slow*, *average*, *fast* and *very fast*). This was used to check that, in the general case, whatever final implementation is used, the basic functioning of the gate is coherent, that is, it is similar to the behavior depicted in Figure 3.7, with no variation of quality in function of the concentration of timer. The second set of parameters was chosen to reflect an integration with the DNA toolbox [67, 11, 43, 119] and is supposed to be more realistic. In particular, we reused the kinetic parameters for strands association, dissociation, and for polymerase activity (reduced ten times to reflect the effect of strand displacement on polymerase activity). Strand displacement kinetics were taken from Zhang and Winfree [61].

The simulation results are shown in Figure 3.5. For all simulation settings (both fuel-driven and fuel-less versions with generic parameters and fuel-driven version with realistic parameters) the gate delays the output for an amount of time linear in the initial concentration of timer before releasing it quickly. As expected, simulated results are similar for all settings, with slightly slacking curves in the case of the “realistic” parameters.

We also simulated the respective effects of a huge excess of fuel, or a lack of it, showing its impact on the system. As shown in Figure 3.6, a lack of fuel completely stops the reaction, rendering the gate useless, but an excess of it slows down the reaction as the $5'$ end of the gate spends more time captured by fuel in step (2) (Figure 3.1). This prompted the development of the fuel-less version, which has the additional advantage of reducing the number of single-stranded species in the solution, reducing the risks of crosstalk with other modules.

Another thing to consider is the release speed of the output once the timer has been consumed: optimally, the concentration of free output should go from zero to its maximum in a very short amount of time in comparison with the characteristic time scales of the elements of

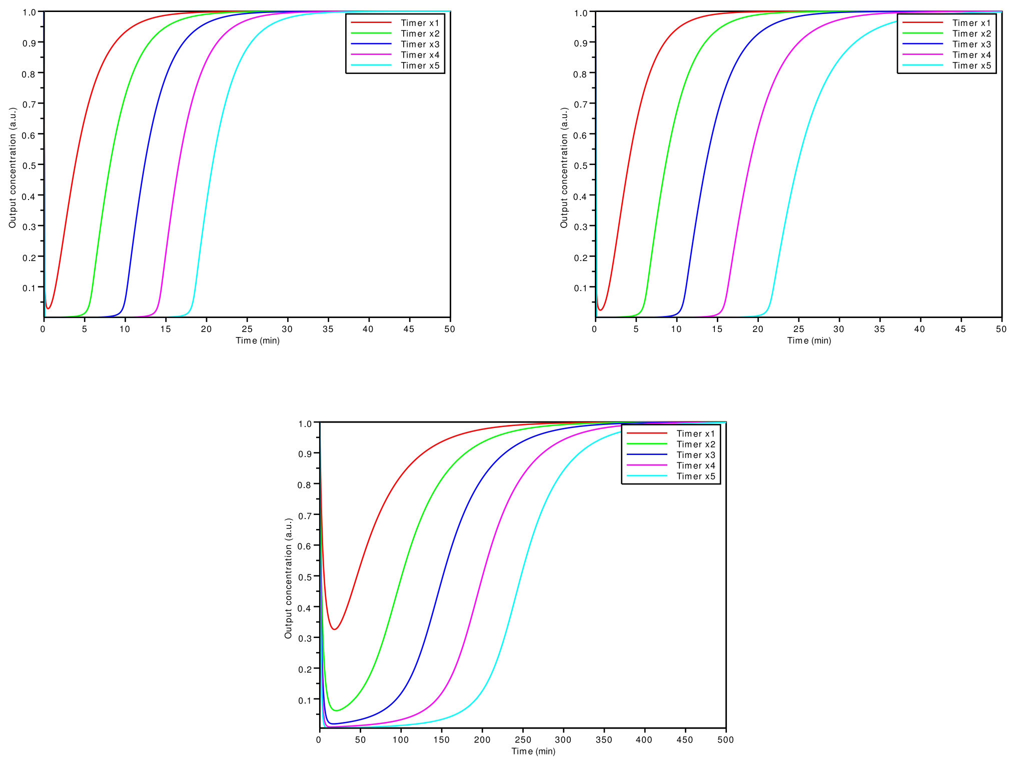


Figure 3.5: Simulation of the delay gate, fuel-powered design (left) and fuel-less design (right) with generic parameters. Simulation of the fuel-driven version with realistic parameters is shown at the bottom. Timer $\times n$ represents a timer concentration n times that of output. Output and gate concentrations were set to 10 nM. Fuel concentration, when applicable, was set to 120 nM.

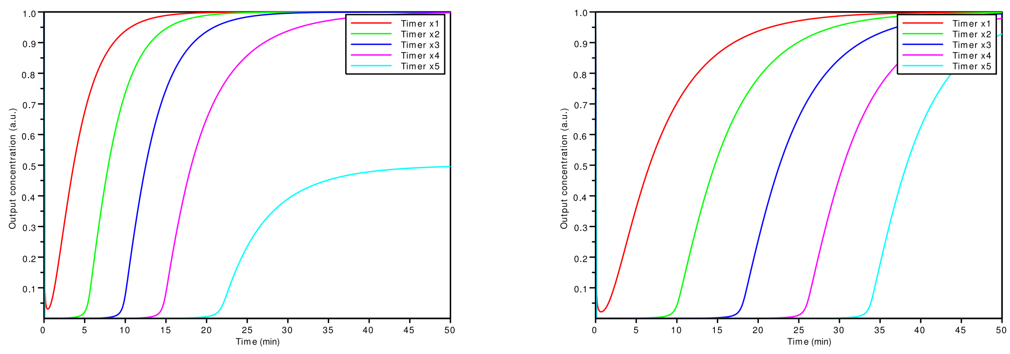


Figure 3.6: Impact of fuel on the behavior of the delay gate. Timer $\times n$ represents a timer concentration n times that of output. Simulations for fuel concentration of 4.5 times that of output (left) and 50 times that of output (right). Output and gate concentrations were set to 10 nM.

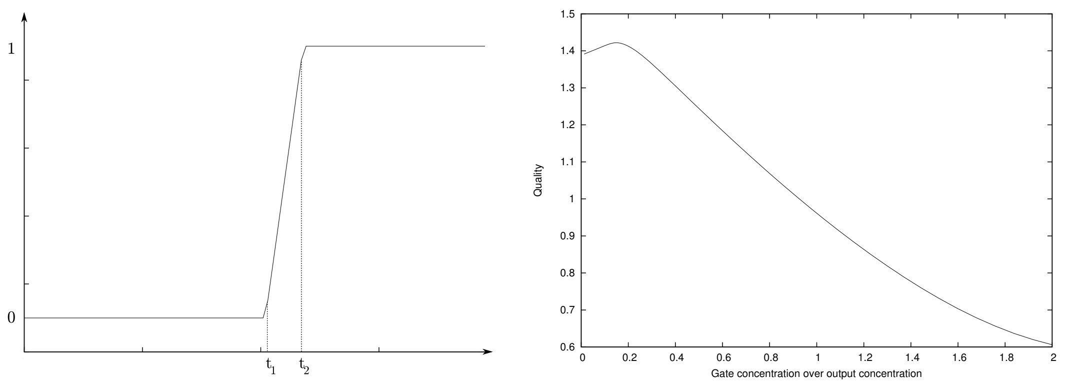


Figure 3.7: Left: idealized behavior of the gate, showing two characteristic times: the delay t_1 and the release time t_2. The quality of the gate $\frac{t_1}{t_2 - t_1}$ depends only of the amount of gate and polymerase present in the solution. Right: quality of the system simulated with generic parameters in function of the concentration of gate, obtained at a timer concentration of $2\times$ the output concentration. The optimal quality is obtained for the gate at $0.15\times$ the output concentration.

the system that the gate is delaying. We also want the gate to be efficient, that is, we want a given concentration of timer strand to delay the output as much as possible. We thus introduce the notion of quality of the gate, defined as $\frac{t_1}{t_2 - t_1}$ where t_1 is the time at which there is 10% release and t_2 the time where the release is 90% complete (Figure 3.7, left). By this definition, the quality is mostly related to the concentration of gate and polymerase in the solution. The concentration of fuel should also be taken into consideration in the fuel-driven version of the gate, but its impact on the kinetics is hard to precisely tune, so in the following we will consider that it is kept at a constant concentration, for instance by using the autocatalytic module of the DNA toolbox [67]. Moreover, the range of possible fuel concentration is limited, since this concentration has to be high enough to power all the gate strands in the solution, while being low enough to prevent it from slowing down the delay system. As the amount of polymerase present in solution is likely to be dictated by the other systems the gate is interacting with, we are left with the concentration of gate as a modifiable parameter. A high concentration of gate means a faster release of the output, but also a shorter delay (that is t_1) for a given concentration of timer, so for a specific release time t_1, more timer and more dNTPs (the fuel for polymerization) will have to be consumed, which leads to a tradeoff, as shown in Figure 3.7, right. In particular, when coupled with consumption intensive systems, such as the bistable system of Padirac *et al.* [11], it may be advantageous to sacrifice quality for the sake of saving energy (dNTPs, fuel, ...).

3.5 Experimental results

While the main purpose of this work was to propose a theoretical generic mechanism to implement delay, we wanted to check the validity of our design. This led us to make an actual in-vitro implementation of the gate, with various parameters, such as buffer conditions, nature of the stopper and so on. In particular, some effort was put into finding the optimal length of toehold that should be captured by the output strand.

In all those experiments, the release of the output was monitored by fluorescence. Output strands had a FAM fluorophore attached to their $3'$ end, while timer strands had a quencher at their $5'$ end, so that only free output would contribute to the fluorescent signal. We made two different implementation of the stopper. Like in whiplash PCR [69], the first one uses a missing dNTP: when the polymerase has to use a dNTP that is not present in the solution, it falls off. The stopper is made of four consecutive occurrences of the complementary of the missing dNTP to guarantee that the polymerase will not 'jump' it. The second option that we considered to make the gate compatible with other systems was to insert a polymerase-blocking modification, similar to scorpion probes [120]. We chose to use a Sp18 spacer, a hexaethylene glycol chain. Since it is not made of nucleotides, the polymerase will not be able to process it and is expected to stop and fall off the gate.

Since the output-timer and timer-gate duplexes are both stable at the working temperature, the competition to hybridize to the n shared bases of the timer strand is a random walk going through all the possible configurations. For this reason, we can approximate that the last $\frac{n}{2}$ bases of the delay gate (see Figure 3.2) are free on average. Using the kinetic parameters from Zhang and Winfree [61], we can see that we need such toehold to be at least 3 bases long for a single-stranded timer to have a reasonable chance to displace the duplexed timer from the gate. Moreover this duplex may have already been partially or totally elongated by the polymerase (*i.e.* the complex output-timer-gate may be at the third step or further). In this case, the only chance to displace it is through the strand displacing activity of polymerase. This means that the single-stranded timer needs to be able to attach stably enough to act as a primer in a reasonable amount of time (that is, before the whiplash had a chance to occur, releasing the output). In a preliminary experiment (see Figure 3.8, left), we found out that there is noticeable leakages if n is inferior to 8. For values superior to 8, the behavior of the delay gate shows little change, except when n becomes big enough to reduce the stability of the output-timer duplex when attaching to the gate. In this case, the release of the output is more difficult, as the output-timer-gate complex is less often in a configuration that would be elongated by the polymerase. The different experimental behaviors for various values of n, in the case of the missing dNTP implementation, are shown in Figure 3.8, left. In this experiment, the relative concentrations of species were output 1, timer 2, gate 1 and fuel 12, with actual concentrations of: output 10 nM, timer 20 nM, gate 10 nM and fuel 120 nM. We can observe that there is little variation in the behavior of the gate for n between 8 and 12, while $n = 14$ shows a noticeable slowdown.

The second thing we had to check was that the release rate of the output was not dependent of the amount of timer initially present in the solution. This is an important factor to determine if gates are left intact after being used, which is crucial for the reusability of this system over time. The experimental result for the missing dNTP implementation can be seen on Figures 3.8, right and 3.9, top left. While the initial amount of timer strands doesn't seem to affect the release rate, we can see that it reduces the total increase of fluorescence, which could be explained by an incomplete release of the output. One possible explanation is that the stopper is not perfect, and gates are slowly degraded, until the reaction grinds to a halt. Similar results are obtained with the spacer implementation, but the release rate is much slower (Figure 3.9, top right). This may be because the whiplash is harder without a toehold directly next to the invaded domain [60]. However there are ways around this problem, such as using LNA [74] to promote the invasion. It is also possible to use artificial nucleotides to stop the polymerase while keeping a proxy for strand-displacement. Finally, we implemented the fuel-less version, as shown in Figure 3.9, bottom. As expected from the model, the release is slower than in the fuel-driven version.

3.6 Applications

As it is, the delay gate is generic and flexible enough to be used in multiple contexts. We present here some of those possible applications, with an emphasis on its integration with the DNA toolbox. This can additionally serve as an example on how to use the delay gate in cooperation with other systems.

3.6.1 Integration of the delay gate in the toolbox

The integration of the delay gate with a given framework takes place in three steps.

The first step is to check the compatibility with the molecular environment. Both the delay

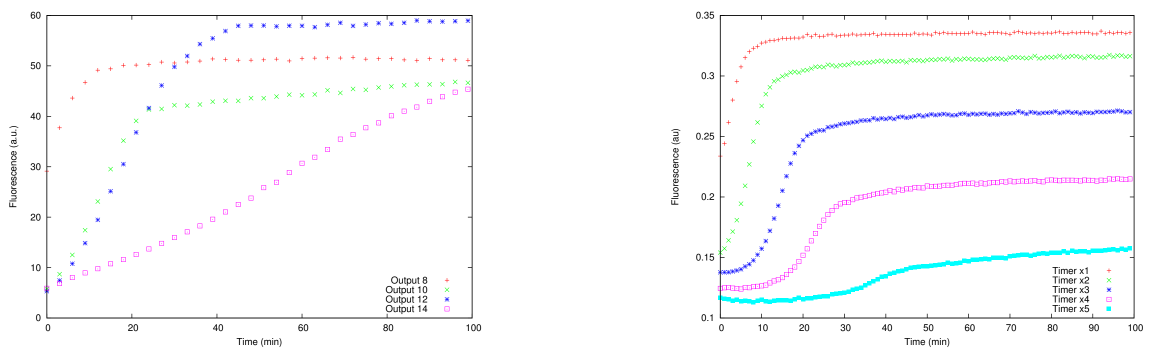


Figure 3.8: Left: comparison of different toehold. Output n corresponds to an output with a cover length of n. Right: experimental results for different concentrations of timer strands, $n = 8$. Timer $\times n$ represents a timer concentration n times that of output. In both experiments, we used Klenow Large Fragment polymerase, diluted 50 times for a final concentration of 100 units/mL. Buffer conditions: NEBuffer 2 (commercial buffer recommended for this polymerase) with dNTP (missing dATP) 100μM. Left experiment was done on Rotor-Gene Q (QIAGEN), right experiment on MiniOpticon (BIORAD).

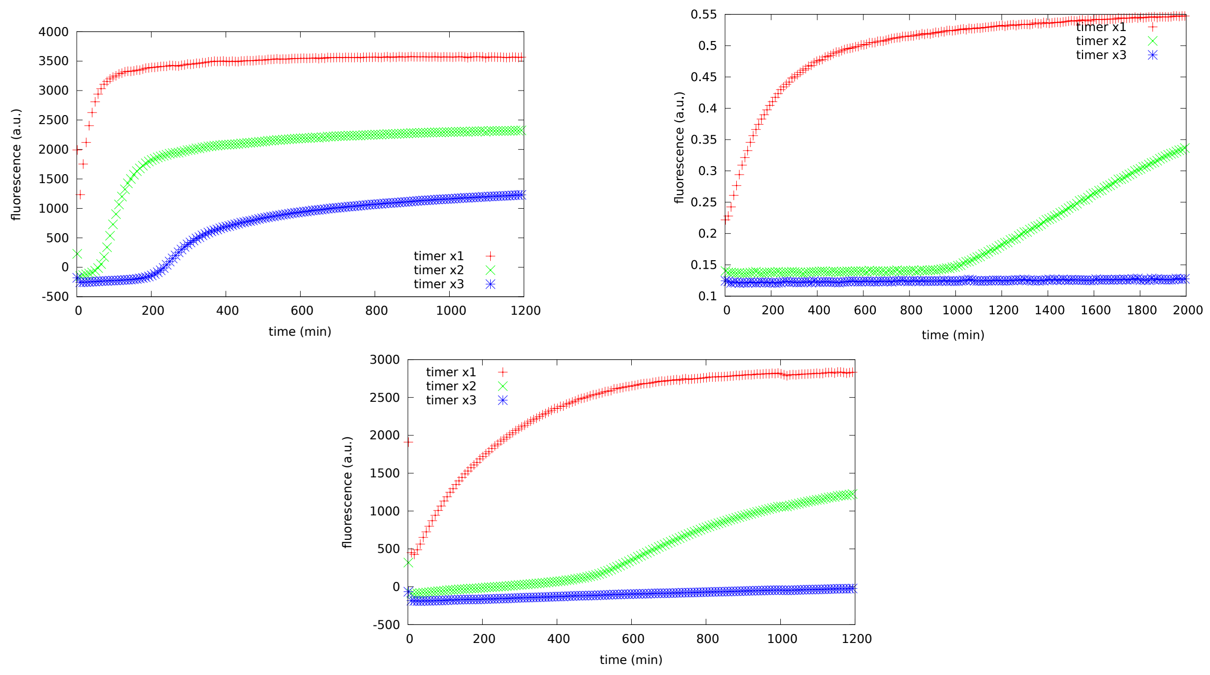


Figure 3.9: Experimental results for different concentrations of timer strands, $n = 12$. We used Bst polymerase large fragment diluted 250 times (final concentration 32 units/mL) to integrate the delay gate in the DNA toolbox (see section 6). Top: fuel-driven versions of the gate. Bottom: fuel-less version. Stoppers are top left and bottom: missing dNTP, top right: Sp18 spacer. Buffer conditions: thermopol buffer (commercial buffer recommended for this polymerase) with DTT 4mM, BSA 10μg/mL and dNTP (missing dATP in the relevant cases) 100μM. Top left and bottom experiments done on IQ5, top right experiment done on MiniOpticon. Note that the time scale of the gate with Sp18 spacer is different. Sequences are given in Appendix A.5

gate and the toolbox use polymerase, and have similar working temperatures. The remaining potential problems come from the other two enzymes: the nickase and the exonuclease. Those problems are easily avoided by respectively avoiding use of the recognition sequence of the nickase and by protecting the $5'$ end of the delay gate against the activity of the exonuclease (see Montagne *et al.* [67] for more details).

The second step is to decide how the delay gate will be used. Since the DNA toolbox was explicitly made to design dynamic systems, it makes sense to plan to reuse the delay gate multiple times. This means that we have either to use the fuel-free version, or to generate the fuel. The later can be assured by a simple autocatalytic module, so ultimately the choice of using one or the other would be case specific, depending on other factors such as how much dNTPs are available and polymerase saturation. For the sake of simplicity, we chose to use fuel-less gates in the rest of this section. Timer generation is done through dedicated templates. We however do not allow sequences from the toolbox to disable/enable the delay gate as of now, as this mechanism uses complementary sequences (see Figure 3.3) and would cause crosstalks between the templates generating the opening and closing strands. The possibility of using a mechanism external to the toolbox for the purpose of disabling and re-enabling the gate is left for future investigation.

The last step is to decide which sequence will be captured by the timer strands. There are two possible targets: signaling sequences or templates. The former seemed limited in its scope as it does not really change the structure of the “program”. Moreover, both activators and inhibitors dynamically hybridize and denature with their complementary sequence at the working temperature. The templates, however, are long enough to form stable duplexes, on top of having the role of defining the topology of the reaction network. As such they are perfect targets for the timer strands. However, since the sequences of templates contain the recognition site of the nicking enzyme, and are, by definition, complementary to many strands present in the solution, some caution has to be taken to prevent crosstalk, as shown in Figure 3.10. An interesting side effect is that removing templates for a given amount of time means that we can modify dynamically the topology of the network: a connection can be removed by generating the appropriate timer, and added when said timer runs out. Let us note that this is not specific to delay gates integrated with the toolbox. It could be possible to control “externally” a specific toolbox system by releasing or capturing some of its templates.

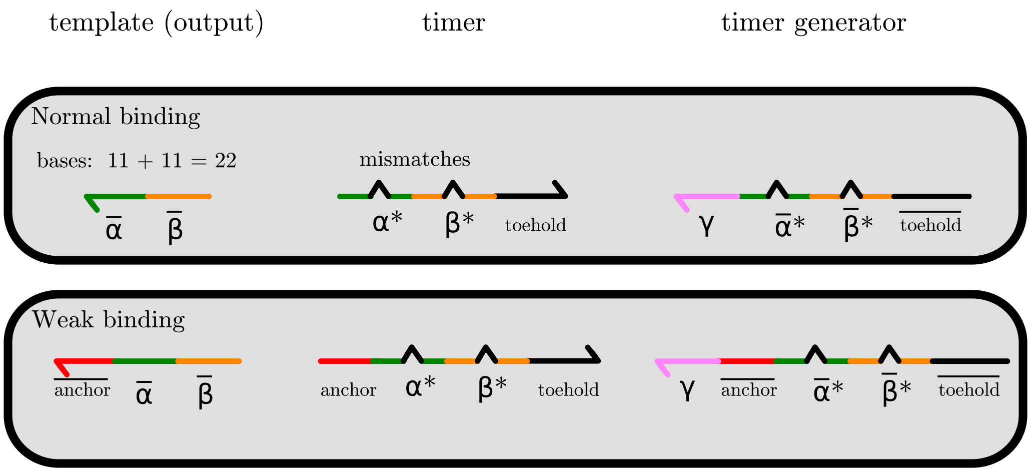

template (output)
timer
timer generator
Normal binding
bases: 11 + 11 = 22
mismatches
$\bar{\alpha}$ $\bar{\beta}$
α* β* toehold
γ $\bar{\alpha}$* $\bar{\beta}$* $\overline{\text{toehold}}$
Weak binding
$\overline{\text{anchor}}$ $\bar{\alpha}$ $\bar{\beta}$
anchor α* β* toehold
γ $\overline{\text{anchor}}$ $\bar{\alpha}$* $\bar{\beta}$* $\overline{\text{toehold}}$

Figure 3.10: Design of a timer strand capturing a template α to β and of the template generating it from γ. A mismatch is introduced in the recognition site of the nickase, and prevent the use of the timer generator as a template α to β. Modified sequences are denoted by an asterisk. Some sequences may be too weak to form a stable duplex with those mismatches. In this case, an anchor sequence may be introduced, as shown in the weak binding design. On the other hand, some sequence may be too stable, and activators may wrongly attach to the timer generating template, in which case additional mismatch are introduced at the position of the activators' $3'$ end. Alternatively, α or β sequences may be designed shorter. There may be more than one toehold sequence, interacting with different delay gates, if multiple templates have to be captured but their release should not be synchronized.

3.6.2 Extended toolbox systems

As a proof of concept we modified some DNA toolbox systems and made new ones.

Montagne *et al.* showed that the occurrence of oscillation requires the introduction of a delay in the reaction network. In their work, this delay is introduced through successive transduction operations (Figure 2.3, left). This oscillator can be easily fitted with a delay gate instead (Figure 3.11, left), which makes it a good first candidate. Using the realistic set of parameters from Section 3.4 along with those from Montagne *et al.*, we could simulate the behavior of such system (Figure 3.11, right). The first thing we can notice about those results is that the oscillations are better defined than those of the Oligator (Montagne *et al.*'s oscillator [67]): the concentration of the reporter sequence goes to zero much faster and the oscillation frequency and amplitude are higher. This can be explained by the fact that the delay induced by the gate is much more efficient than using intermediate transduction. Additionally, it becomes possible to program the generated delay by using signal strands opening (respectively closing) the delay gate, thus inactivating (respectively reactivating) the third step of the gate (Figure 3.1). This would allow some other mechanisms (based on DNA Strand Displacement for instance) to interface with and control the oscillations. Those characteristics make the delay gate-based oscillator a promising first candidate for a wet implementation of a delay gate system.

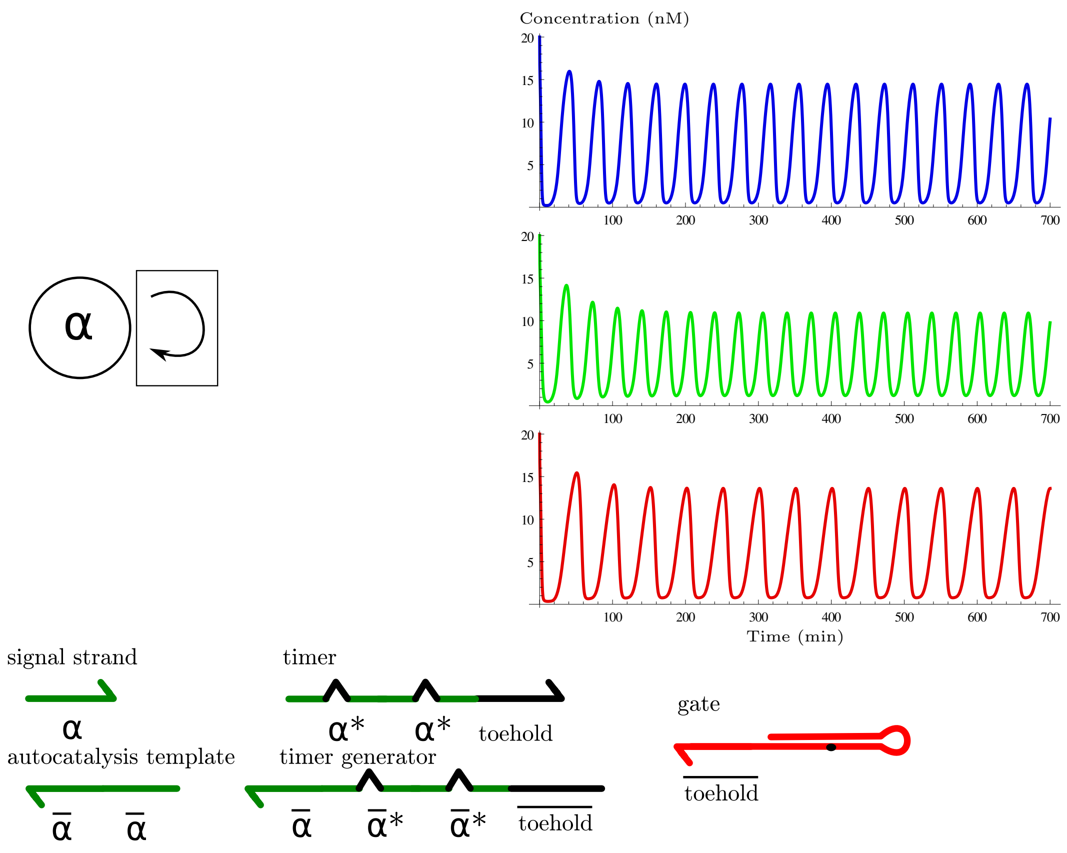
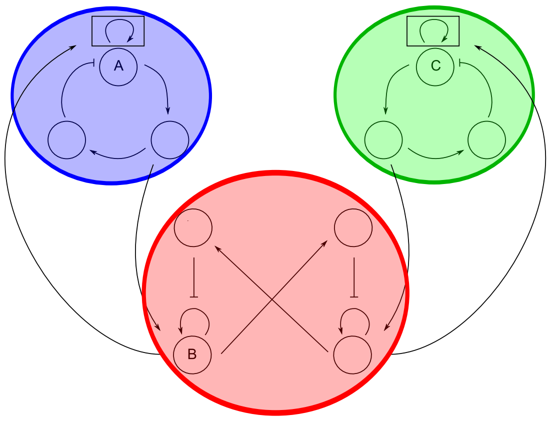


Figure 3.11: Top left: the modified Oligator with the delay gate depicted by a box. Strand α activates the creation of timer for the delay gate, represented by the activation arrow going to the box. This timer will capture the template used for the autocatalysis. Top right: simulated concentration over time of this system. From a given behavior (blue), one can change independently the concentration of timer generator to change mostly the amplitude (green) or change the concentration of gate to change mostly the frequency (red). Bottom: the single-strands encoding this system (enzymes not shown).

To find inspiration for a new kind of system to implement with the delay gate, we turn toward concurrent systems algorithms, since they specifically require join operations. We chose to implement a simplified version of the producer/consumer problem in which multiple producers (respectively consumers) are trying to increment (respectively decrement) a common stock. The state of the stock should always be kept valid (no out of bound or negative values, no jumps), which has been proven to be equivalent to allowing only one actor to modify it at a time [114, 121]. In our version, we have only one producer and one consumer, both implemented using the standard Oligator. They also have an additional delay gate each to be set to sleep while the queue is updating. The queue itself is implemented using Padirac *et al.*'s bistable circuit, which means it can contain at most only one element. The complete network of the system is shown in Figure 3.12, left. It works as follow: first, the bistable circuit is initialized in the state meaning "empty". In this state, it continuously produces the timer strand for the consumer Oligator. After one oscillation, the producer Oligator switches the state of the bistable circuit. While it is switching, both "empty" and "full" states are competing, which cannot be considered as a valid state. However, this also means that timer strands for both the producer and consumer are generated, preventing them to access the queue. Once the bistable system reached its new stable state, the situation is reverted: the producer is now set to wait while the consumer can make one oscillation. Therefore we obtain alternating single cycles of the two Oligators. This system can also be extended by cascading multiple bistables modules and make a n-bits counter (see next Chapter), which would also require the addition of a mechanism for both the "add one" and "remove one" operations. Those operations are feasible using the toggle switch also presented by Padirac *et al.* [11]. The simulated time trace of this producer/consumer system can be seen in Figure 3.12 on the right. As depicted, the oscillators are never accessing the queue (oscillating) when they are forbidden to, that is when the bistable hasn't reached a stable state or when trying to remove (respectively add) something from an empty (respectively full) queue.

3.6.3 Other possible applications

There are multiple possible usages for delays. First, we can see in Figure 3.12 that the system has to go through an initialization phase where the states of the oscillators and bistable are meaningless. This comes from the fact that, at the beginning, we cannot create a valid state since we have to start all the subparts of our system, which requires to add sequences that are forbidden to be present in non-negligible concentrations at the same time. Specifically, in our



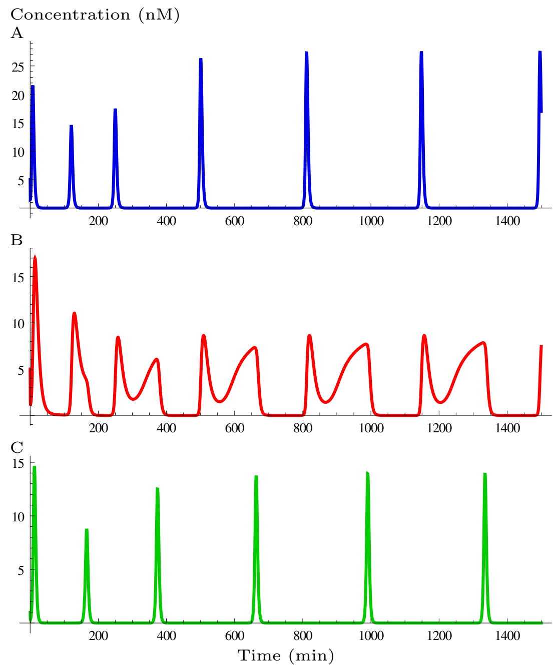


Figure 3.12: Producer/consumer problem. Both Oligators are designed to have different periods to avoid fake synchronization. Left: the network implementing the system; the producer is blue, the consumer green and a status of the stock is red. Right: the simulated time traces of the A (producer), B (full state) and C (consumer) sequences, following the same color scheme.

producer/consumer, the consumer needs a first oscillation to start, even though this oscillation is actually meaningless. Luckily, the simulation was able to revert to a valid state, but this may be harder in real life. This problem can be generalized to many complex systems where all mutually exclusive subparts have to be initialized at the beginning. The solution would be to delay some strands until the state of the system is valid. In the case of the producer/consumer problem, for instance, the starting sequences for both oscillators could be delayed until the bistable had reached a stable state. Let us note that the delay gate used for initialization doesn't have to be from the extended DNA toolbox, that is, it could use different timer, toehold, fuel sequences and possibly a different stopping mechanism, and is not subject to the same compatibility requirements presented in the previous Section. In the producer/consumer example, however, we can use the gates that are already present in the design, and delay the autocatalytic templates of both oscillator by adding the appropriate timer strands.

Another use of the delay gate would be to synchronize systems. If two strands are set to wait with timers using the same gate (meaning that they have the same toehold), both outputs will be released at the same time, regardless of which was set to wait first or the relative concentration of timer of each.

Finally, the gate can be used to “hide” parts of the system that are present in the solution. This allows a system to dynamically modify itself, a behavior that is reminiscent of other learning strategies implemented with DNA systems [122]. In particular, this could solve the problem of the addressable space of sequences: in the previous Chapter, we mentioned that to prevent crosstalks, sequences have to be different enough, which limits the diversity of possible individual sequences. In turn, this restriction affects the maximum system size, as at some point it is not possible to add any new species without having crosstalks. However, if only subsets of the system were running at a given time, this size limitation would be lifted, as all subsets could use the same set of sequences safely. This is identical to context swapping in "regular" computers: a given memory address will represent different things based on which process is currently running. Note that this mechanism only limits crosstalk among the sequences that are directly delayed by the delay gate, the rest of the system still being subject to this problem. However the advantage is that potential crosstalks are limited to a much smaller part, which should make designing and checking much easier. For instance, in the producer/consumer problem, it is possible to use similar sequences for both oscillators, the design difference with the system in Figure 3.12 being that whole parts of the system (all templates in the colored circles, plus the templates representing the interactions with the bistable circuit) have to be delayed. Timer strands relative to a specific oscillator have to be synchronized to a common delay gate to prevent different parts of said oscillator from being released asynchronously. The next point is that, if the sequences of the oscillators are similar, their respective timer strands are going to be similar as well. This problem is solved by introducing enough mismatches in the respective timer strands to make them only stable with their cognate targets. This means that one delay gate will not wrongly be delaying the other part of the system. It is even possible to do so in the case of completely identical sequences, by adding an "addressing" area at the $3'$ end of the templates (similar, for instance, to Soloveichik *et al.* [6]). In this last case, there will effectively be two "copies" of the same oscillator, locked in different phases. The addressing space has the additional advantage of increasing the stability of the timer-template stability, which might be low due to the large amount of required mismatches. Another possibility to increase this stability is to use modified nucleotides with higher melting temperatures in the template, such as LNA [74].

3.7 Conclusion

In this Chapter, we presented the design and realization of the delay gate, a specific module dedicated to implement a delay/join operation, while being both autonomous and reusable. It relies on a modified version of the whiplash PCR to ensure reusability and toehold mediation kinetics to ensure the correctness of the release time. It comes in two flavors: a fuel-powered version and a fuel-free version. While the fuel-powered version is less convenient to use, it has the advantage of better characteristic release times. The delay gate was also designed to be controllable either by changing the concentration of timer strand (useful for systems capable of generating arbitrary concentration of sequences) or by disabling its own whiplash mechanism (which is more targeted toward entropy driven systems which can only generate fixed amounts of strands, for instance by releasing them by toehold exchange). This allows the delay gate to be compatible with multiple DNA computing systems, such as DNA Strand Displacement and the DNA toolbox, or even to work as an interface for such systems, allowing to create hybrid systems. Finally, the delay gate can mitigate crosstalks by preventing direct interactions between parts of the system. While this mitigation is system-dependent, it still helps by offering additional design options.

We checked the feasibility of such module by doing an actual *in vitro* implementation, along with some parameter optimizations. We then showed how we could add the delay gate to an existing framework, using the DNA toolbox as an example.

What is left is to find a good implementation of the delay gate, correcting the output release problem. We suspect that this problem is due to the stochastic nature of the stopper we use. Correcting this design is thus mostly a (time-consuming) matter of finding the appropriate molecular modification for the job. Once the gate works with little to no signal loss, it is important to create the systems we presented *in vitro*. Finally, we can note that the delay gate helps us with combining small systems that we already know, such as Montagne *et al.*'s Oligator [67] or Padirac *et al.*'s bistable circuit [11], but we are still far off from being able to combine DNA toolbox modules in arbitrary ways. In the next Chapter, we will present a software tool to automatically generate systems from their graph, allowing a much finer control.

Chapter 4

DACCAD: a Computer-Assisted Design approach for the DNA toolbox

In the previous Chapter, we advocated the use of new building blocks to help design DNA computing systems. The delay gate we introduced can be used either with the DNA toolbox, or other paradigms. However, there is a limitation to this strategy: even if, eventually, it gets simpler to combine advanced modules, there is no point if we loose more time designing those modules. Like in the assembly codes used by processors, it is much more important to have a reasonable set of instructions that can be combined easilly. “Easilly” is key, as it means that we can delegate this operation to the computer, thus writing a compiler from a human friendly, high-level representation, to a descriptive, efficient, low-level one. This latter representation can then be used, for instance, for precise simulation or to design the actual DNA sequences needed for an *in vitro* implementation.

Additionally, the number of parameters included in these simulations can be large, especially in the case of complex systems. For this reason, we included the possibility to use CMA-ES, a state of the art optimisation algorithm that will automatically evolve parameters chosen by the user to try to match a specified behaviour.

Finally, since all possible functionality cannot be captured by a single software, DACCAD includes the possibility to export a system in the Synthetic Biology Markup Language, a widely used language for describing biological reaction systems.

We described the low-level model we use for the DNA toolbox in Chapter 2. The graphical representation of DNA toolbox systems is much nicer to manipulate for a human designer. We created DACCAD (DNA Artificial Circuits Computer Assisted Design) a graphical user interface that allows us to do so. The results presented in this Chapter have been submitted as reference [97].

4.1 Introduction

Computer Assisted Design (CAD) has helped many fields reach their full potential by scaling up designs, making error-prone or repetitive tasks automated and allowing easy simulation and verification of systems. As such, CAD has proved to be a necessary tool to develop any advanced technology, examples of which include areas as varied as integrated circuits, aeronautics or programming. The case of DNA computing, a field that uses the interactions between various DNA molecules and enzymes to perform meaningful operations, is no exception: while DNA computing systems have shown great promises [10, 41, 67, 11], advances are still considerably limited by the trial and error process, and many such systems are often left at the proof-of-concept stage. Computer assistance could remove this limitation and help designers take the next step toward actual applications by helping them implement complex CRNs (Chemical Reaction Networks).

CRN are assembly of cross-interacting chemical reactions where each molecular compound can affect the formation and degradation rate of the others. CRN, when kept out of equilibrium, can display a variety of nonlinear behaviours, including oscillations and multistability. They are very powerful systems in terms of information processing, as shown by the variety of complex cellular processes that are controlled by CRN *in vivo* (see, for instance, Wagner and Fell [123]). The recent demonstration that CRN possess Turing universality [10, 6, 8, 16, 9] also advocates their use in a variety of control tasks at the molecular level. However, CRN tend to resist rational design. A link between the properties of the network and its dynamic functions can be obtained from mathematical tools such as dynamical systems theory, including linear stability analysis [99], Lyapunov stability analysis [124], or CRN theory [125], but such analytical approaches are generally restricted to idealized systems described by very simple mathematical models; actual implementations are generally much more intricate in terms of chemical kinetics.

As we noted in the first Chapter, great efforts have been invested in the exploration of reaction networks relying on *in vitro* DNA chemistry and biochemistry. These systems are based on the predictability of DNA-DNA interactions using simple Watson-Crick base-pairing rules, as well as on the rich repertoire of possible enzymatic transformations. Using DNA-based molecular programming, it is possible to build simple computing systems based on boolean logic [6, 126, 127] but also neural networks [41] and CRN [8]. Enzymatic systems are of particular interest because they can be maintained out-of-equilibrium for an extended period of time (for instance by having large amount of dNTPs or NTPs in the solution acting as a generic fuel) and thus are suited to explore emergent behaviours. We focus on the DNA (Dynamic Networks Assembly) toolbox, a recent framework to assemble out-of-equilibrium networks of arbitrary topology, whose dynamics are powered by three enzymes (polymerase, exonuclease and nicking enzyme). This toolbox has been used to build oscillators [67], bistable systems [11], a push-push memory circuit [11], or even molecular ecosystems reproducing predator-prey cycles in bulk [119], spatially [128, 43] or in droplets [129]. Spatial implementation of those systems display waves [128, 43], showing they can be used as a base for two-dimensional reaction-diffusion systems. Most of these systems were simple enough to be designed and tuned by hand, but doing so was still an arduous process of trial and error that cannot be extended to larger systems. Indeed, in well-mixed systems, any molecular element can potentially interact with many others at any given time, and keeping track of all of these interactions and their kinetic consequences quickly becomes impossible for a human. It gets worse when counter-intuitive effects, such as competition and saturation [12, 103, 89], have to be taken into account.

However, a quite unique feature of DNA as a molecular material is, that it is possible to derive simple rules and kinetic parameters describing its behaviour in the general case [8, 61, 130], complexity emerging from the non-linearity of those mechanisms. Similarly, the biochemistry of DNA can be described by simple models, such as the Michaelis-Menten equation or competitive inhibition. Therefore DNA-based CRN can be both implemented in test tubes and be described by quantitative or semi-quantitative ODE models based on standard kinetics. Compared with previous experimental studies on CRNs, the availability of these exact mathematical models is the true benefit of DNA systems. It puts the molecular programming approach in stark contrast with "classic" nonlinear reaction networks, such as those observed in chemistry (like the Belousov-Zhabotinsky reaction [20]) and in biology (gene regulation networks [131, 132], signalling cascades in metabolic networks [133], etc...). In these last two cases, indeed, while it is generally possible to infer a global pattern of interaction among the molecular members of the reaction network, it is much more difficult to obtain precise mathematical forms describing these interactions, let alone accurate numerical parameters.

A DNA-based system thus provides the chemical tools to implement any computation or dynamic behaviour at the molecular scale. This implies the construction of complex systems whose behaviour quickly becomes intractable for the human brain. Such systems require non-linearity, be it through signal amplification, digitalisation or the generation of autonomous dynamics such as oscillations. The biochemistry of DNA systems provides such mechanisms, but assembling them in a constructive manner is still a difficult and sometimes counter-intuitive process. Moreover, realistic prediction of the actual evolution of concentrations over time requires a number of side reactions, such as leaks, crosstalks or competitive interactions, to be taken into account. In this case, the design of a system targeting a given function requires many trial and error before the correct architecture can be found. To speed up this process, we have created DACCAD, a Computer-Assisted Design software that supports the construction of systems for the DNA toolbox. DACCAD is ultimately aimed to design actual *in vitro* implementations, which is made possible by building on the experimental knowledge available on the DNA toolbox [67, 11, 43, 100, 101] (see Chapter 2). Because of the existence of these detailed mechanistic models, it is still possible to numerically simulate such systems with reasonable accuracy, justifying computer assisted design. For those reasons, we expect that this approach will be the only way to push the complexity of experimentally accessible systems further. In DACCAD, the services offered by the machine are fourfold:

- It provides a straightforward graphical interface allowing to create the network and assess its dynamic behaviour.
- It eliminates routine and error prone tasks such as ODE writing and solving.
- It helps in simple optimisation tasks such as adjusting the parameter to tune the behaviour of the network toward a quantitative target, once a qualitative agreement has been obtained.
- It can display an animated version of the network, giving a better understanding of the evolution of the system over time.

Moreover, DACCAD was written entirely in Java to make it as easy to deploy as possible. This software allows the user to set the reaction parameters, such as enzymatic activities, to be as close as possible to real life experiment. It also comes with preset parameters from Padirac *et al.*'s work so that only a general knowledge of CRN is necessary to start designing complex systems. DACCAD thus makes it quick to test designs based on the user's intuition. If the resulting behaviour is not the one expected, DACCAD also incorporate tools to both detect where the problem might be or optimize such behaviour toward a target. Users can also refer to the supplementary materials of Montagne *et al.* [67] to fit the parameters (enzyme activities, strands dissociation constants) to their own experimental settings. Designed systems can then be built *in vitro*, using Baccouche *et al.*'s protocol [101]. Finally, DACCAD can convert any such designed system to SBML to ensure compatibility with other simulation tools that the user would like to use. It is also possible to export the core model to Mathematica (options like enzymatic decoupling or leaks are unavailable, but it is then possible to add delay gates to the system).

We illustrate its effectiveness by designing various systems, from Montagne *et al.*'s Oligator [67] or Padirac *et al.*'s bistable system [11] to new and complex networks, including a two-bits counter or a frequency divider, as well as an example of very large system encoding the game *mastermind*. This is one of the advantages of the DNA toolbox: in a generic biochemical simulation program such as COPASI [134], the user would need to add all those species and reactions by hand, setting the kinetic rates of each reaction based on enzymatic saturation. DACCAD generates all those elements by implicitly referring to a precise biochemical context, allowing the user to design systems much faster. Generating a template takes one click in DACCAD, while it generates for COPASI 5 new species, 11 reactions and requires to update carefully the Michaelis-Menten equation of each enzyme. It gets even worse for COPASI if the template is inhibited, as an additional species and 6 new reactions have to be added to the mathematical model. Figure 4.1 shows the size difference between the representation of the same reaction network in DACCAD and COPASI. In this Chapter, we highlight a variety of behaviours, such as enzymatic saturation and load effect, which would be hard to handle or even predict with a simpler model. We also show that those mechanisms, while generally seen as detrimental, can be used in a positive way, as functional part of a design.

4.2 Related work

The basic operations that can be executed by DNA, such as two complementary strands attaching to form a double helix, toehold-mediated strand displacement and enzymatic transformations, are well-known. However, there are multiple paradigms that use those mechanisms in very different ways. One such paradigms, DNA Strand Displacement (DSD), already benefits from its own software for computer-assisted design, VisualDSD [96], demonstrating the effectiveness of the CAD approach. Recently, Kwiatkowska *et al.* were able to prove the incorrectness of

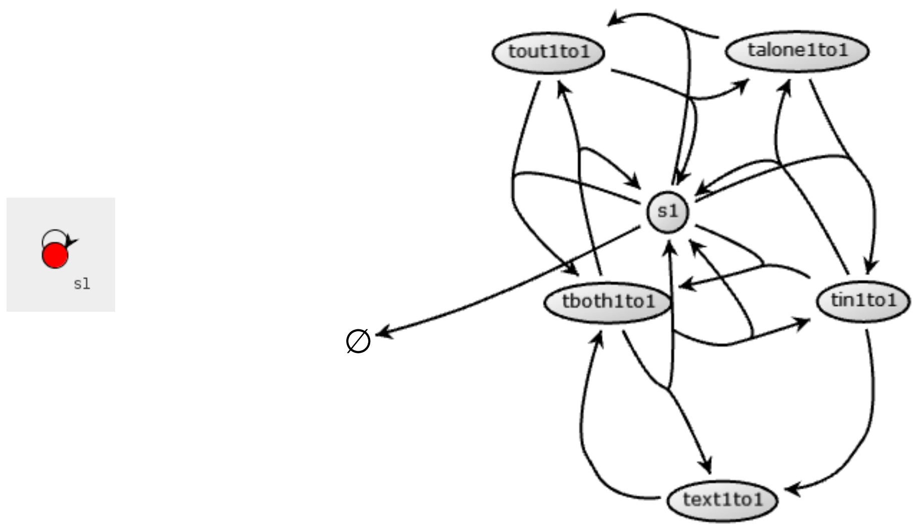


Figure 4.1: Comparison between the representation of a simple DNA toolbox autocatalitic module in DACCAD, and the same chemical system in COPASI. The latter graph was created through the interface of COPASI from the SBML file exported by DACCAD.

a basic DSD system by coupling VisualDSD with a verification software and then propose a corrected version [102]. Visual DSD is also useful to highlight counter intuitive phenomena such as the winner-take-all effect [12, 118, 42].

DSD is also the foundation of Qian and Winfree's seesaw gate [10, 41], which they used to build complex logical circuits. However, the translation is hard to do by hand, and so is the design of the actual hundred of DNA sequences making up the system. This is why they designed a compiler[1] which takes as input a description of a logical circuit and outputs the DNA sequences that would implement it. The compiler can also export the system in the Mathematica and Synthetic Biology Markup Language (SBML) [90, 91] format for simulation of the system at the molecular level, and Visual DSD format for simulation at the toehold level.

The reason for designing automatically DNA sequences in DNA systems is that unwanted interactions at the molecular level are very common. While simulation might show no problem with abstract sequences, actual implementation, if done without care to the potential crosstalks, can lead to unexpected secondary structures. In particular, since sequences are made up of only four nucleotides, it is hard to prevent partial complementarity. NUPACK [93] and DINAMelt [57, 94] are on-line tools designed to check how specific sequences will interact together and output the most stable configurations. They also compute many characteristics of such structures, such as melting temperature. NUPACK focuses more on the secondary structures of systems with multiple kind of DNA strands, while DINAMelt provides various data on the interaction

[1]http://www.dna.caltech.edu/SeesawCompiler/

of two sequences.

For broader applications, COPASI [134] is a very complete tool for analysis and simulation of biological systems. It has the advantage of being able to perform mass action, stochastic and hybrid simulations of any system described in SBML and is also able to handle parameter fitting and optimisation, which makes it very powerful. However, it has the inconvenience of requiring the user to provide the SBML file describing the system, or alternatively input each and every single molecules, interactions and reaction rates by hand through the graphical user interface, which is a long process (by human standards) for realistic systems. For this reason, like the seesaw gate compiler, we gave the option to export systems designed with DACCAD in the SBML format so that the user can perform additional operations with COPASI, or any other software supporting this format.

Furthermore, DNA possibilities are not limited only to computation. Since DNA molecules can be considered flexible when long enough, but locally rigid, they can be used to build complex structures, for instance by using a long scaffold strand held in place by short “staple” strands [135], or by using directly specific short strands [29, 55]. Such approaches have to rely heavily on CAD software, such as caDNAno [136] and CanDo [56], as the number of different sequences necessary to create complex structures is very large.

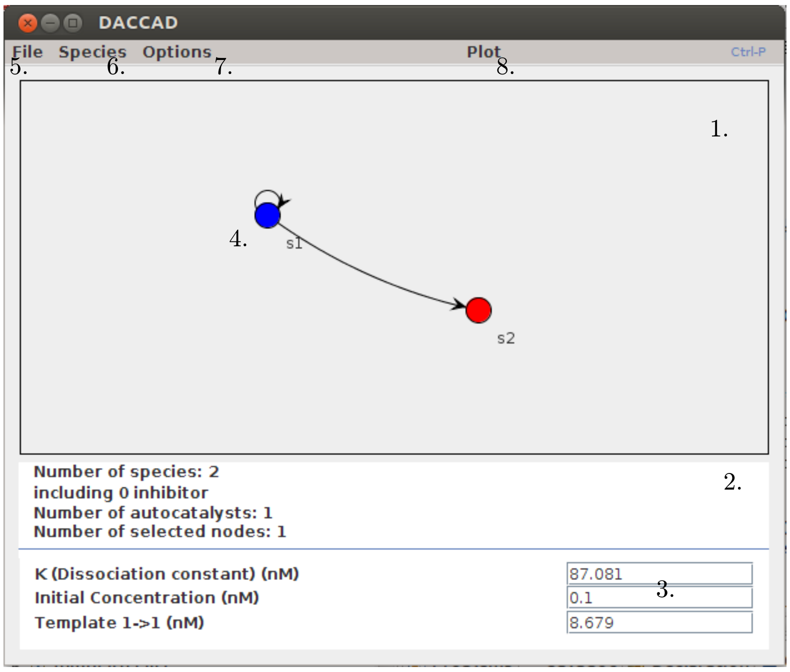
Also note that those different pieces of software work at different levels, ranging from abstract species in COPASI to DNA strands in VisualDSD or DACCAD to actual DNA sequences in NUPACK or DINAMelt, depending on the application.

4.3 Methods

4.3.1 Graphical interface and graph manipulation

The main graphical interface of DACCAD is separated into two areas (Figure 4.2): a display panel showing the graphical representation of the system being designed and a data panel showing context-dependent information.

As mentioned before, DNA toolbox systems can be represented as a graph, with a minor modification to allow inhibitors to target templates. The Graph class from the JUNG library [137] was extended to store this information, as well as additional sequences and template-relative parameters, such as stability and concentration. The display is done by the JUNG library, with a modified renderer that is able to draw inhibition arrows as well as changing the colour of selected nodes. A second renderer was created for the graph animation, to support



Number of species: 2
including 0 inhibitor
Number of autocatalysts: 1
Number of selected nodes: 1

K (Dissociation constant) (nM) 87.081
Initial Concentration (nM) 0.1
Template 1->1 (nM) 8.679

Figure 4.2: **1.** Display panel. **2.** Data panel. **3.** Parameters area. Sets sequence and template-relative parameters.**4.** Graph of the system. Selected nodes are displayed in blue. **5.** File menu. **6.** Sequences menu. **7.** Options. Sets general and enzymatic parameters. **8.** Plot. Simulates and display the behaviour of the system.

size and colour changes. The plotting of time traces is done by the JFreeChart library [138]. Most graph operations (adding/removing a species, duplicating group of nodes and so on) have a simple equivalent in terms of impact on the DNA system. The join operation is the only one which is not completely straightforward. It is done by a simple flooding algorithm exploring all the currently selected nodes to detect autocatalytic cycles. The species resulting from this operation will have an autocatalytic template if and only if there was at least one autocatalytic cycle (that is, a cycle in the directed graph of all selected species). All inhibiting strands targeting a template (a directed edge in the graph) taking part in such cycle are fused, since they are all inhibiting the same autocatalytic cycle. The fused inhibitor species is created and set to target the autocatalytic template of the new signal strand. All templates inbound toward the selected strands are redirected to the new species. Similarly, all outbound templates are set to initiate from it.

4.3.2 Default parameters

Association constants and strand-displacement speeds are based on Zhang *et al.*'s values [61]. Strands dissociation constants and Michaelis-Menten parameters for the enzymes are taken from Padirac *et al.*'s measurements [11]. The value of enzyme activities are also taken from Padirac *et al.*'s experiment, except for the nickase activity, for which the fitted value was taken instead. In simple cases, saturation doesn't modify much the behaviour of the system, which is why saturation can be disabled. By default saturation is enabled, both to be closer to the actual *in-vitro* behavior and to help design systems specifically using this phenomenon, like in a winner-take-all system [12, 118, 42].

Realistic values for coaxial-stacking were hard to define, as there is not only a strong sequence dependence [98], but this dependence is not even limited to the nearest-neighbours [104]. To get our values, we considered the opening/stacking energy described by Frank-Kamenetskii's group [105]. We then applied salt and temperature (42°C) corrections [106] to match the experimental values of the DNA toolbox. This yields a range of values between 0.04 (25 times slowdown for a nicking site GC) to 1 (no slowdown for a nicking site TA). The various possibilities are listed in Appendix 4 and are provided as a pop-up in DACCAD. The default value corresponds to a five times slowdown, but can be changed to match the actual sequences used in a particular experimental setting. Note that there are many different and sometimes contradictory measures of coaxial stacking, so DACCAD can only provide a best effort approach.

4.3.3 Equations generation and solving

In Chapter 2, we have explained how to reduce a given graph to a system of differential equations. This system is actually never explicitly generated by the simulator. Instead, rewriting rules are applied to deduce all the flux affecting a given part of the system on the fly, generating the value of the derivatives of the various parts of the DNA system. Those derivatives are passed on demand to the Gragg-Bulirsch-Stoer integrator [139] from the Apache Commons Mathematics Library, generating the time trace. This algorithm was chosen for its high precision and step adaptability, useful to solve systems such as the bistable switch [11] where two excluding parts of a system get very close in concentration and requires careful integration to get the correct dynamics.

The system of differential equations is only actually generated for export as a Mathematica or SBML file. A generic file template containing the basic rules of the DNA toolbox chemistry is completed through similar rewriting rules to the one used for the simulator.

4.3.4 Local optimisation

While trial and error is a simple way to optimise *in-silico* a system's behaviour, such as an oscillator's amplitude or frequency, it might be inconvenient due to the sheer number of parameters existing in the model (duplex stability of every species, initial concentrations, template concentrations, Michaelis constants of enzymes). For this purpose, we added the possibility to use CMA-ES, a state-of-the-art optimisation algorithm [92], to perform parameter optimisation. The structure is kept as defined by the user to make optimisation possible in a reasonable amount of time. Structure optimization can also be attempted, but requires adapted algorithms [140, 141]. The user can define which parameters can be optimised, which is useful to accommodate existing constraints or reduce computation time.

The desired behaviour is defined by placing dots on the time plot of concentrations or by loading a previously saved target profile. CMA-ES will then try to minimise the least square distance between the target and the actual concentration curves. If no match can get below the user-defined termination threshold, the algorithm stops after a given number of iterations (default is 100) and set the parameters to the best fit. Note that this fit may actually be worse than the original behaviour from the human perspective —such as a flat concentration curve equal to the average concentration of the targeted oscillator— or better, but not good enough. In those cases, it is possible to undo the changes or ask for another round of optimisation.

4.3.5 Dynamic graph display

Linking structure and behaviour of a toolbox system can be challenging, especially with large systems. For this reason, we implemented a module displaying a dynamic representation of the network. This module provides an animated version of the graph, in which the radiuses of the nodes are updated to be proportional to the logarithm of the concentration of their corresponding species at a particular time. We chose to use a logarithmic representation to be able to see variations of concentrations at multiple orders of magnitude. The edges are also updated, using a gradient of colour to represent the concentration of corresponding templates being occupied by a signal strand (i.e. a completely inhibited connection would appear in grey). Furthermore, to make the graph easier to read, it is drawn using a standard spring-electrical graph drawing algorithm [142] (each node is given a repulsive force for all other nodes and an attractive force which applies specifically to the nodes with which it is connected). This has proven to be satisfying in terms of display results and computing time.

In the last version of DACCAD, enzymatic saturation is also displayed, giving the user a better understanding of the under the hood mechanisms going on at a given time.

4.4 Results

In this Section, we present various systems that were simulated using DACCAD, going from the most simple to advanced networks. We also present the effects of enzyme saturation and show that they can be both deleterious or instrumental, depending on the context. The actual designing process is presented in the next Chapter.

4.4.1 Simple systems

As a proof of concept, we implemented Montagne *et al.*'s Oligator [67] and Padirac *et al.*'s bistable circuit [11]. The time series were compared to those of their original article. An additional reason to do so is that those systems represent two basic behaviours, respectively oscillations and memory, making them appropriate building blocks for advanced systems. Multiple time plots, showing the impact of different parameters on those systems are shown in figure 4.3. This also gives a chance to quickly explore the impact of multiple parameters on such systems. In particular, coaxial stacking (and thus indirectly the actual nucleotide sequences) has a strong impact on autocatalysts. For the Oligator, the system gets more stable oscillations if the autocatalyst has little to no coaxial slowdown. In the case of the bistable circuit, it is possible to find a stability diagram based on the initial concentrations of A and B similar to the one found by Padirac *et al.* [11]. This diagram is strongly influenced by the coaxial stacking values of the various templates. Based on the actual sequence used by Padirac *et al.* and their corresponding stacking energy as defined by [106], it was possible to explain the unusually high stability observed in one of the autocatalysts, showing that not only the overall stability, but also the nucleotides at the nick have to be taken into account when designing DNA sequences. Polymerase affinity for particular sequences [109] might also change the relative strength of one side or the other, but can be approximated by a slight increase of the relevant template concentrations.

4.4.2 Simple system optimization

The Oligator only oscillates on a restricted area of the complete parameter space [67]. To find a working set with specific properties in terms of amplitude and frequency, we used the optimization tool of DACCAD, leading to the results shown in Figure 4.4. In this particular case, a very good match was found, but specific behaviours may be impossible. In case of troubles with the evolution, it is recommended to reduce the number of evolved parameters

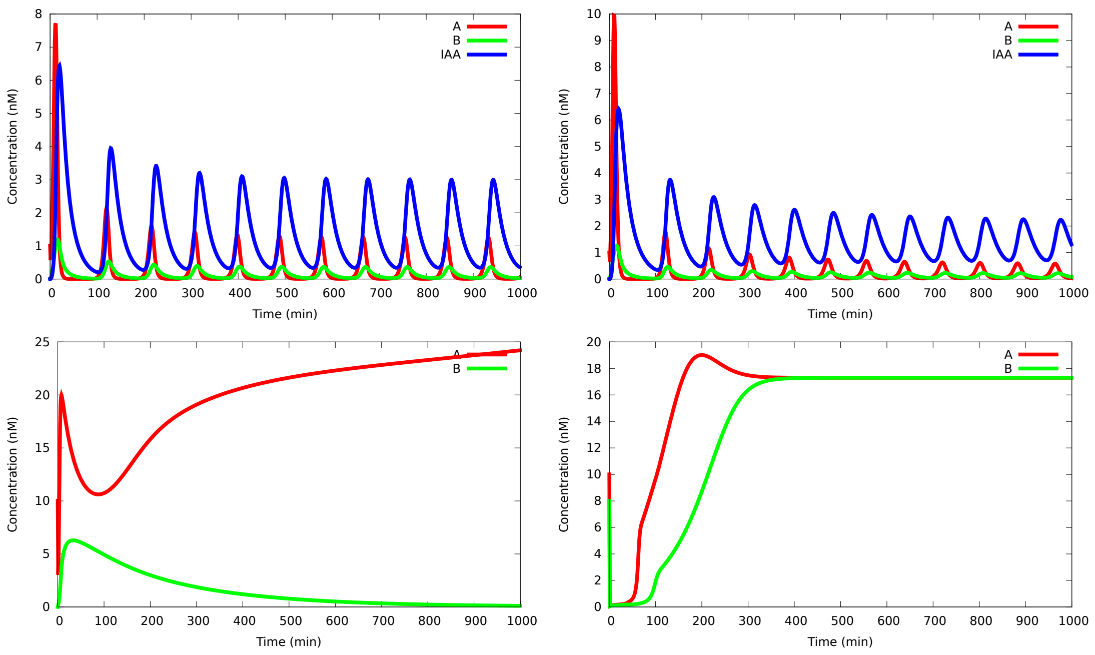


Figure 4.3: Top: the Oligator [67], simulated with first-order enzymatic activity (left) and with enzymatic saturation and competitive coupling(right). With the chosen parameters, the amplitude and frequency of oscillation is modified, but the behaviour is otherwise unchanged. Bottom: simulation of Padirac's bistable circuit. Only the time plot of the autocatalytic species is shown. The initial concentration of A is higher than that of B, forcing the system into the A state. If the concentration of autocatalytic templates is too high (right), the system saturates the enzymes and loses its bistable properties. Performing some optimisations on the enzymes parameters gives us some insight on this phenomenon: it turns out that with Padirac *et al.*'s parameters [11], the nicking enzyme saturates first, causing this loss of bistability.

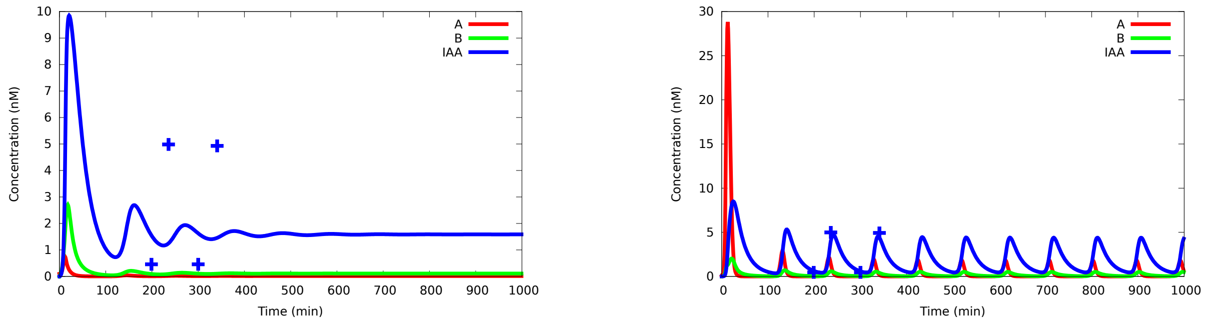


Figure 4.4: Optimisation of an oscillator. At first, the system only displays damped oscillations. The user can describe the desired behaviour through the interface (left, the dots represent the user-defined behaviour). CMA-ES then performs some rounds of optimisation until a result close enough is achieved (right). Note that the concentration scale is different between the two figures.

and go through multiple intermediary behaviours so that the system evolves progressively in the good direction. As an example, when trying to optimize the frequency divider (Figure 4.7), the optimizer was not explicitly allowed to modify the behaviour of the oscillator. However, by increasing the concentration of the template connecting the oscillator to the rest of the circuit and using the load effect, the optimizer found a "good" match: the additional templates sequesters an intermediary species of the oscillator, making it actually twice slower. Good parameters were found by excluding this parameter from the optimization as well.

4.4.3 Animated display

Padirac's switch oscillator uses three autocatalitic modules inhibiting each other in a circular pattern. Interestingly, the oscillations occur in the reverse order of inhibition (Figure 4.5), a fact that might be counter-intuitive. It can be hard to find this property by only reading the time plot, if one is not looking for it. On the other hand, the activation order is obvious when watching the graph animation.

4.4.4 Combined systems

Using the two previous building blocks, we can build advanced systems that would require hundreds of lines if the ordinary differential equations were written by the user. By cascading two bistable systems, we can get a two bit counter (Figure 4.6). If instead we chose to combine an extended Oligator and a bistable, we can get a frequency divider (Figure 4.7). The systems presented here are perfect examples of the problems that might arise when combining two circuits. While the modularity of the DNA toolbox makes it simple to use multiple systems in parallel, like in the two bits counter, many factors have to be taken into account when cascading

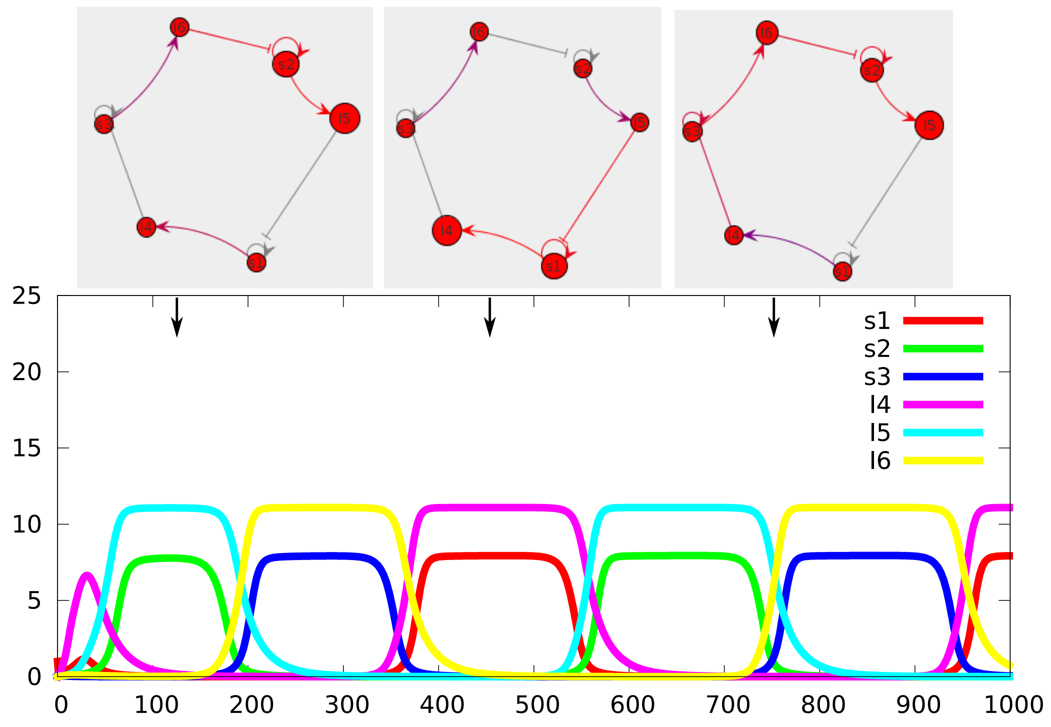
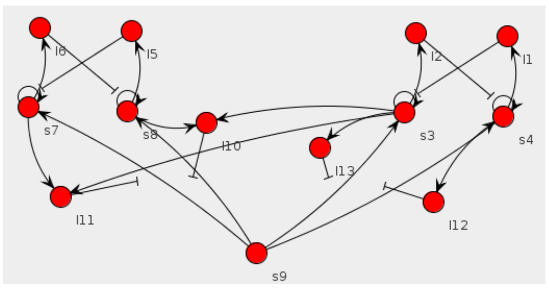


Figure 4.5: Graph animation, relative to the time trace. The simulated system is a three-switch oscillator [43] without enzymatic saturation.

systems. The most important is the load effect: when adding a template taking a given strand as input, the sequestering of this strand will change the behaviour of the system it is part of. In the frequency divider, for instance, the template connecting the oscillator to the bistable system will change the oscillator's frequency or even prevent anything more than damped oscillations. This phenomenon is strongly dependent on the strand used as input: any perturbation to the autocatalyst will prevent true oscillations, but the intermediate species are more robust. The optimal course of action is then to create a link to a new species that will in turn bear the full load of the connection to the other part of the system.

4.4.5 Saturation-based effects

Kim *et al.* [12] presented a biochemical phenomenon named winner-take-all that arises when there is competition for a common resource in which the species that reproduces the fastest using this resource will eliminate all competitors. This phenomenon was leveraged by Genot *et al.* to perform efficient computation [118]. In the DNA toolbox, enzymes are perfect example of such shared resources [42]. Since the standard experimental parameters are set to avoid saturation (and thus competition) as much as possible, the activity of one enzyme has to be



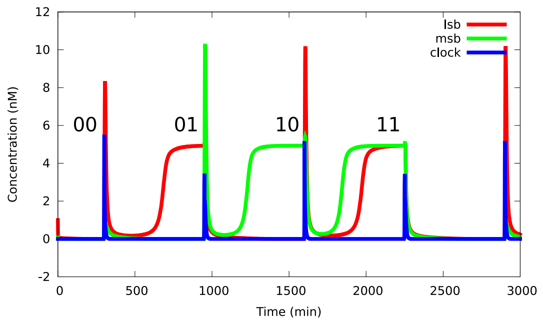
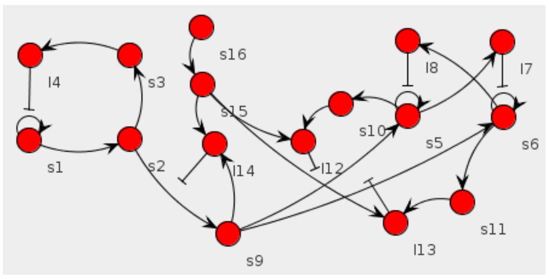


Figure 4.6: A two bit counter made of two push-push memory circuits [11], with the actual circuit of the system shown on the left and the time plot of the relevant species shown on the right. Switches come from an input of species 9 (clock), making this a non-autonomous system. The system goes through the cycle species 8 (most significant bit, msb) low/ species 4 (least significant bit, lsb) low; species 8 low/ species 4 high; species 8 high/ species 4 low; species 8 high/ species 4 high. Note that the spike of species 9 when switching the least relevant bit is different when going from 0 (species 3 high) to 1 (species 4 high) than when going from 1 to 0. This is due to the load effect: when species 3 is high, more templates that can capture species 9 are inhibited, so the free concentration of species 9 is higher.



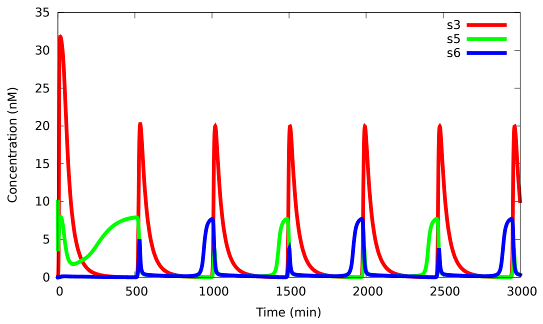


Figure 4.7: A frequency divider. The original oscillator (s_1 to I_4, a version of Montagne *et al.*'s Oligator [67] with a two species delay) is connected to a push-push memory (s_5 to I_{11}). The two species delay has the double effect of making the oscillator more robust to the connection load and slowing the oscillations to give enough time to the bistable circuit to switch. To produce spike-like activation of the switch, an additional species (I_{12}) is used to form an incoherent feedforward loop. Species s_{13} and s_{14} are used to initially inhibit the switch, giving enough time to the bistable switch to reach a valid state. Both state species from the bistable can then be used as oscillators of half the original frequency, in opposition of phase.

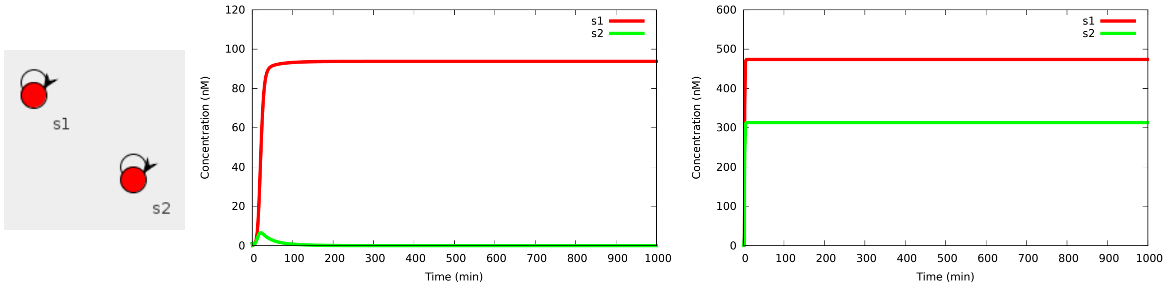


Figure 4.8: Winner-takes-all effect. Time trace of two simple autocatalysts, with (center) and without (right) saturation. Interaction is only done through competition for the polymerase enzyme, other enzymes are set to work in first order regime. In this case, the autocatalyst s_1 has more template than s_2, resulting in the complete disappearance of the latter.

reduced by an order of magnitude. This can be achieved in an actual experiment by reducing the actual quantity of enzyme introduced in the system. Each of the three enzymes creates different behaviours when they are set to be the bottleneck of the reaction system, leading to interesting dynamics. For instance, very low exonuclease means that a dominating species will receive most of the degradation, thus protecting other species that would not survive otherwise. Conversely, very low polymerase enables the winner-takes-all effect described by Kim *et al.*, as shown in Figure 4.8. Saturation can also be harnessed to generate various behaviours, such as oscillations (Figure 4.9) in a system that would be stable otherwise.

4.4.6 Complex system: the *mastermind* game

Finally, as a proof of concept of modular design, we implemented a large system where repeating motifs were connected together. For this purpose, we chose to implement a version of *Mastermind*, a classic board game in which one player decides of a secret combination that the other player tries to guess. A combination is a sequence of symbols (usually colours) in which order is important. After each guess from the second player, the first player announce how many symbols are correctly placed (correct guess) and how many symbols among the remaining do appear in the combination, but at a different position (misplaced guess). A formal description of those rules is given in the next Chapter. While, in the original game, there are multiple possibilities for each position, we simplify it to use only two different symbols (A and B). For this reason, we also don't give the amount of misplaced guesses, as it would make the game too easy.

We designed a simple version of this game using DACCAD. In our version, we use four bistable motifs to store the secret code decided by one player. The opponent may inject one of two possible species per position. If the correct species was selected, a downstream species

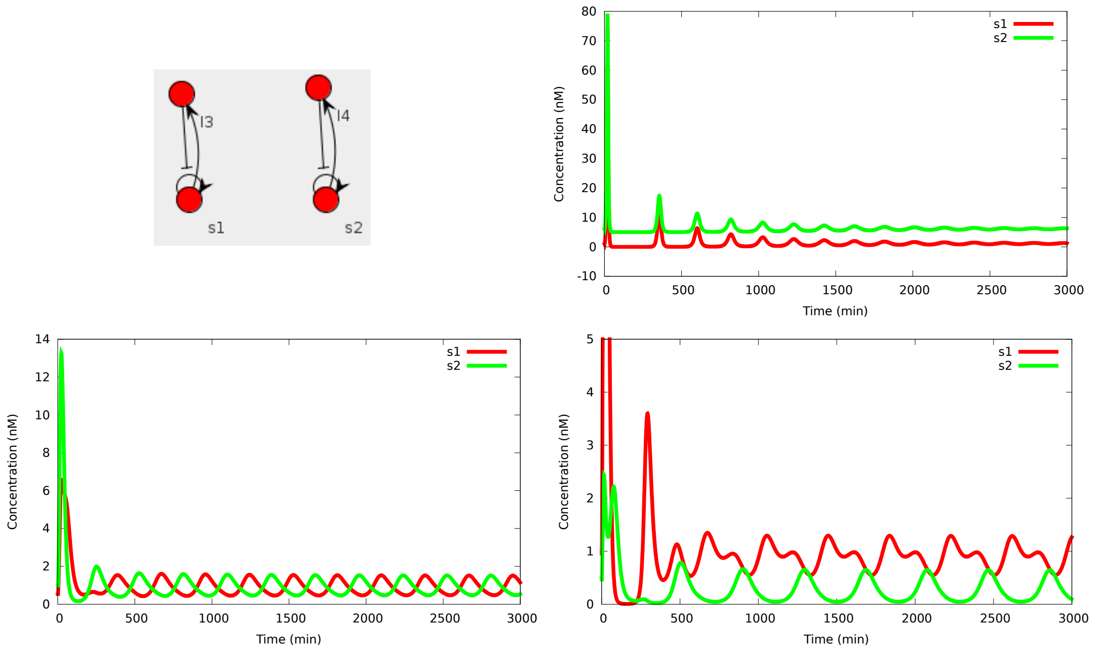


Figure 4.9: Saturation-based oscillations. In those systems, saturation works at the polymerase level. When one autocatalytic module starts saturating most of the polymerase, its concentration will increase, but, at the same time, so does the concentration of its inhibitor. On the other hand, the inhibitor of the "loosing" autocatalytic module will be produced slower and slower, while being degraded by the exonuclease. Once the current winner is inhibited enough, the tendency are inverted and its concentration starts decreasing while the other autocatalytic module starts sequestering the polymerase. Without saturation, the system reach a steady-state after damped oscillations (top right). Note that the green oscillator has been translated upward to prevent the overlap of both curves. On the other hand, when both short Oligator are competing for resources, an oscillatory regime appears. Oscillations can be symmetrical if templates concentrations are identical (bottom left), or asymmetric if there is a small difference in concentrations (bottom right).

is activated that we call “reporter” species. If all “reporter” species are present, then the concentration of the “lock” species falls to zero, indicating victory. The whole system is shown in Figure 4.10. To determine the number of correct guesses, the concentration of the reporter species may be measured through fluorescence in a real implementation. By using the same fluorophore for all of them, the only relevant information is the level of fluorescence which is equivalent to the number of correct guesses. It is also possible to use the drop of the “lock” species as a hint, although this modification is not linear with the number of correct results, which might be confusing.

The design of this system was done step by step, starting from a single bistable circuit (blue subsystem in Figure 4.10). This circuit was then complemented to prevent the expression of the reporter species (yellow in Figure 4.10). Then the two possible inputs were added. DACCAD was not only useful to adjust the various template concentrations until the behaviour was correct, it also showed that inputs strands were not surviving long enough to have an impact on the system. This prompted the development of the specialised input module (green in Figure 4.10). The indirect activation generates a large amount of intermediary species (see Figure 2.4), which in turn has enough strength to allow the creation of reporter species (yellow in Figure 4.10). Finally, a direct activation is also added to compensate the delay induced through the indirect activation.

Once the complete reporter circuit was considered correct, it was duplicated and the two resulting circuits were connected, forwarding information about the correctness of the guess. Then this whole system was duplicated once more and connected through an inhibition of the lock species (red in Figure 4.10). Template concentrations were then adjusted so that the behaviour of the system, that is the level of the lock species and reporters reflected roughly linearly the amount of correct guesses. Particular attention was given to the case with only two correct guesses, which can result in two possible configurations: either both guesses are on the same “side” (that is on two directly connected bistable circuits) or on the opposite sides. In the first attempts at creating the system, there was a strong asymmetry between those two cases, due to the fact that the first case would saturate the template going from the connection species (like s_{42} in Figure 4.10) to the final inhibitor, so that inhibition was stronger when distributed. This was solved by adjusting concentrations so that the production of both connection species would always stay in the linear response zone of the templates connecting them to the final inhibitor. A complete description of this process is given in the next Chapter. A sample game is shown in Figure 4.11.

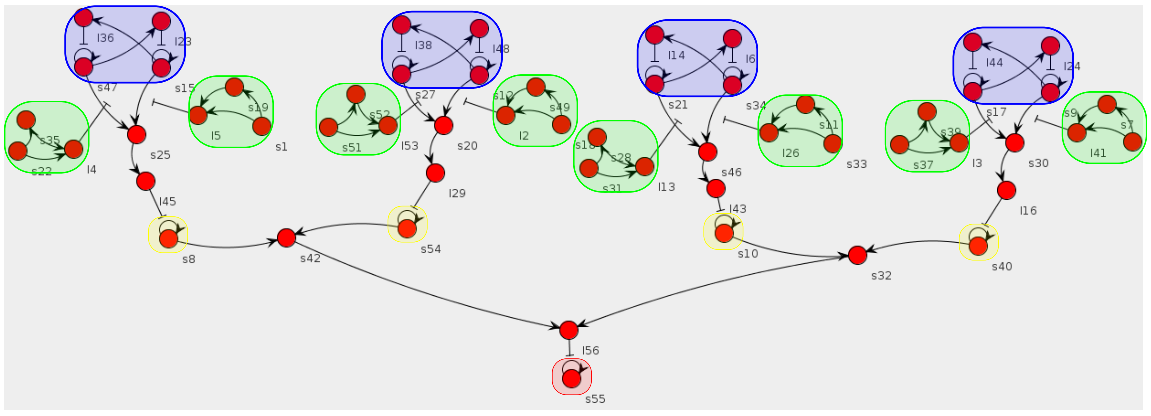


Figure 4.10: Implementation of the mastermind game. State bistables are shown in blue, possible inputs in green, reporter species in yellow and lock species in red.

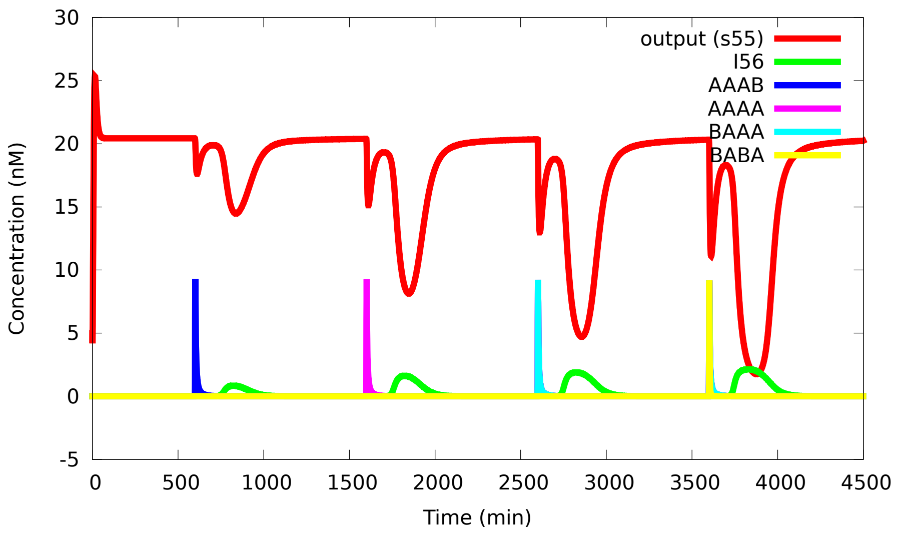


Figure 4.11: Mastermind game. The state of a given position can be either A or B. Each attempt improves the guess until the correct answer, BABA, is found on the fourth trial.

4.5 Conclusion

In this Chapter, we presented a mathematical model for the simulation of systems designed with the DNA toolbox. We also introduced DACCAD, a software created to help design such systems. DACCAD allows its user to quickly create new DNA toolbox systems by using its intuitive graphical interface. Such systems can then be refined at will by setting a large range of parameters to fit the user's experimental conditions. Once the design is complete, it is possible to move on to the actual DNA sequence design, using tools such as NUPACK [93] or DINAMelt [57, 94] (see Baccouche *et al.* [101]). While the value of some parameters, such as the release slowdown due to coaxial stacking, can be hard to predict, the actual value can be measured through some basic experiments and then updated in DACCAD, allowing to quickly correct a design. Such parameters can also be slightly modified from their actual experimental value to encompass more specific or sequence-dependent effects such as the different affinity of the polymerase for different sequences [109]. Finding a way to fit such experimental parameters directly from fluorescence data could be an important future application. Additional operations can be performed through external software as DACCAD models can be exported in the widely used SBML format. In particular, it is possible to quickly generate the DNA toolbox part of a system and then move on to another software to add operations that are more trackable from the human designer's point of view. In that sense, DACCAD can be seen as a compiler that takes a graph format and turns it into SBML, allowing the algorithmic generation of DNA toolbox systems (for example, systems trying to solve instances of the 3-SAT problem [143]). Moreover, this generation is facilitated by its simple graph description file format. DACCAD can also be used to generate Mathematica files, using the core model of the DNA toolbox. Such files have a built-in compatibility with the delay presented in the previous Chapter, so that delays can be easilly introduced in the system.

We then showed how the software can be used to design systems of increasing complexity. In particular, by judiciously setting the parameters, one can also observe different behaviours, such as saturation-based oscillation or the winner-takes-all effect. Local optimisation can also be performed by using the state-of-the-art algorithm CMA-ES. Moreover, the mathematical model can be easily reused with other design techniques, such as evolutionary algorithms, which may help optimising systems following multiple possible objectives [144], or find new design patterns to create complex systems [140, 141]. It would also be interesting to investigate two-dimensional reaction-diffusion implementations of the DNA toolbox [128, 43], which would prove beneficial to develop smart materials or create particular spatio-temporal patterns. To do so, DACCAD could be extended to add diffusion terms to the current equations, and then export them to be solved by off-the-shelf reaction-diffusion simulation software, such as ReaDY [2].

It is our hope that this software will speed-up the creation of novel networks, as well as spread the usage and concepts of DNA computing and molecular programming to a broader audience. The mathematical model of the DNA toolbox presented here might also give new theoretical insights on its power, such as whether it would be possible to approximate arbitrary CRN. The executable file and all examples from this article can be found online[3]. The code can be obtained by contacting the author.

[2]code.google.com/p/reaction-diffusion
[3]hagi.is.s.u-tokyo.ac.jp/~nathanael/cadtoolbox.html

Chapter 5

Using DACCAD to create complex systems

In this Chapter, we will explore two different approaches to create complex systems with DACCAD (DNA Artificial Circuits Computer Assisted Design). The first part will focus on creating a system "by hand", using all the tools existing in DACCAD. We will use the Mastermind game from the previous Chapter as an example. This choice is justified by the fact that this system required to use almost all of the options available in DACCAD, showcasing the possibilities of our program.

The second part will focus on the scripted generation of systems. This last approach relies on known patterns, such as oscillators or bistables. Those patterns can then be quickly assembled, following some generation rules, and as such this process can be considered a compilation from one model to the DNA toolbox. The rational for this approach is that such compiler would help designers with very little biological background create impressive applications. Hopefully, with the help of experiment automation [55], DNA computing systems will have a much broader audience in a near future.

5.1 Creating the Mastermind game

The rules of this game were briefly introduced in the previous Chapter. As a reminder, this game is played by two players with an asymmetrical role. The first player makes up a secret combination. Formally, this combination is a string of symbols $s_1 s_2 \ldots s_n$, with the set of possible symbols Σ finite and known by both players. The second player must then try to guess it in the least amount of tries possible. For each guess $g_1 g_2 \ldots g_n$, the first player gives a hint, in the form of the number of correct positions and misplaced symbols in the guess. A position m is considered correct if and only if $g_m = s_m$. Among the remaining (incorrect) positions, the number of misplaced symbols is the number of symbols that could be correct if they were presented in a different order. Formally, it is equal to

$$\max_{\sigma \in O} Card(\, \{g'_i | g'_i = s'_{\sigma(i)}\} \,)$$

with O the set of all permutations over the set of incorrect positions, g' (respectively s') the maximum substrings of g (respectively s) striped of all correct positions, and $Card$ the cardinality of a given set. For instance, if the combination was red-green-blue-black (symbols being colors), a guess of red-red-red-red would give one correct position, zero misplaced symbol and a guess of black-blue-green-red would yield zero correct, four misplaced.

The following will describe step-by-step the creation of a DNA toolbox system implementing those rules, including the multiple design choices that were made along the way.

5.1.1 Encoding the secret combination

The first step to design this system is to define the encoding of the secret combination. Indeed, this game requires a way to store information, as the system will check multiple time against the player's attempts.

There are two theoretical approaches to store data: passive and active. The passive approach stores data in a stable media which can be read at a later time. In the DNA toolbox, only the templates are stable over time, so encoding would be done through the initial concentration of a given set of templates. It could be also possible to use components "outside the (tool)box", such as the DNA tweezers [31], Seeman *et al.*'s DNA actuator [32], or even our delay gate (see Chapter 3) used in its *join* configuration.

The active approach relies instead on producing continuously some species corresponding to the saved value. This is the case of Padirac *et al.*'s bistable circuit [11], for instance. Additionally, the pattern they used is not limited to bistability and can be easily extended to n-stability (a 3-stable system is shown in Figure 5.1). It has the main drawback of requiring a constant creation of signal, which adds to the burden of the various enzymes.

However, the active approach has two majors advantages over the passive one. First, the bistable circuit was well studied by Padirac *et al.* [11, 43], which makes it a safer bet for an actual *in vitro* implementation with other DNA toolbox components. Second, it is possible to switch

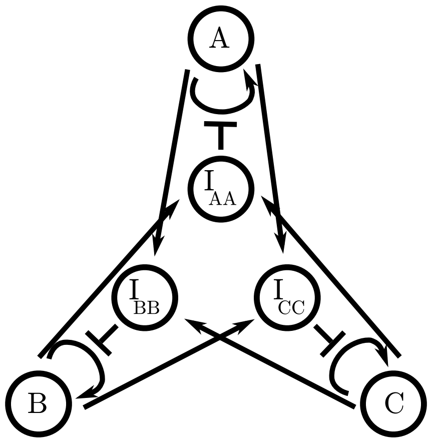


Figure 5.1: The structure of a 3-stable circuit. Each autocatalytic species represent a possible state. The stability is guaranteed by the inhibition of the other autocatalytic modules. We can note that increasing the number of stable states requires a quadratic increase of the number of templates.

the state of a bistable circuit on the fly, meaning that we could reset the game during a single experiment. Conversely, the proposed passive systems are either outside the DNA toolbox and may have some compatibility problems (DNA tweezers, actuator), have only partial software support (delay gate) or cannot be updated dynamically (templates alone). Note, however, that Genot *et al.* showed that it could be possible to update template concentrations by using DNA Strand Displacement systems as a complement [42]. Additionally, the delay gate is supported by the saturation-free model and may be a valid design option for a smaller system.

However, a n-stable system requires n autocatalytic templates, plus $n \times (n-1)$ inhibition templates, for a total of n^2. It is much more efficient, in theory, to use multiple bistable circuits to represent symbols, as we can store n states using $\lfloor \log_2(n) \rfloor$ circuits. Moreover, this does not take the enzymatic burden into account. A n-stable circuit produces n species (1 autocatalyst and $n-1$ inhibitors), while the group of bistable circuits produce $2 \times \lfloor \log_2(n) \rfloor$ species. This means that, overall, binary encoding is the best. We then encode each position of the sequence by a group of bistable (enough to encode all possible symbols).

5.1.2 General design

Based on the previous observation, we then chose to create a Mastermind game with only two symbols and four positions. Since binary encoding is optimal, any version of the game with more symbols would be a direct extension of this design with multiple bistable circuits combined to represent one position. The choice of four positions is arbitrary and can be easily modified.

We decided to use a divide and conquer approach for the logical evaluation of a guess from the player. Each position in the guess will be attributed a dedicated subsystem that will

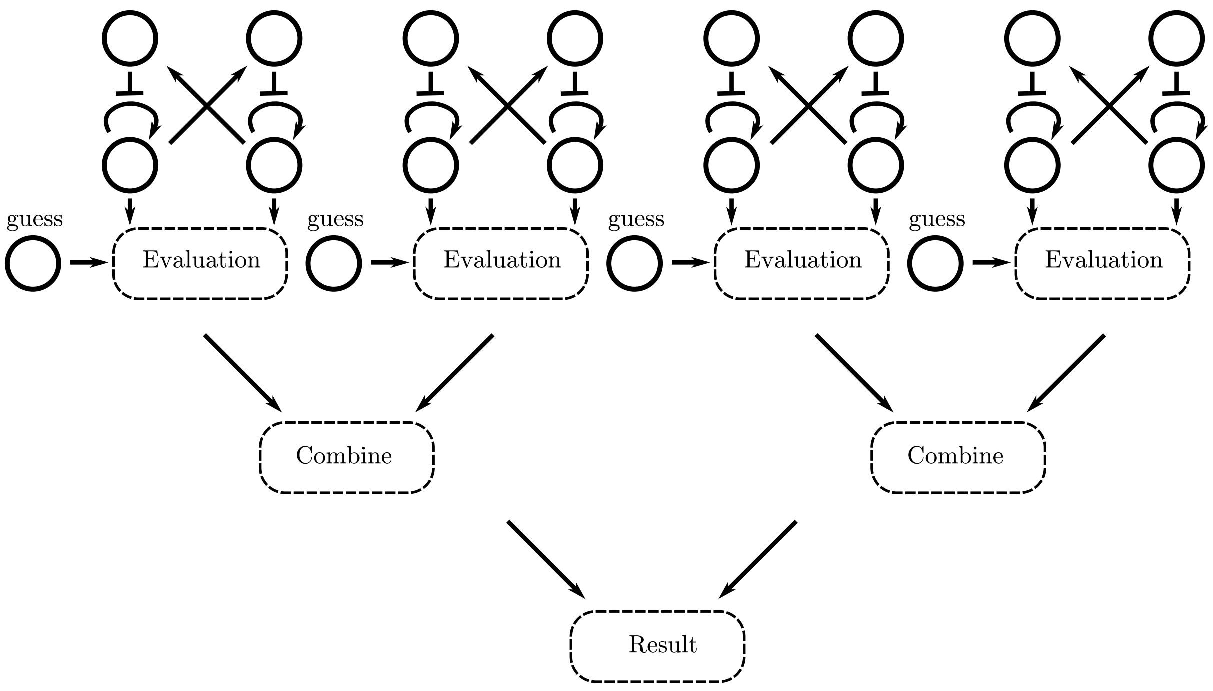


Figure 5.2: Drafted design of the decision system using the divide and conquer approach.

evaluate its truth value. Results are then grouped two by two until they percolate to the top. Molecular computing is very efficient for this approach, since the analogue signal created by the concentration of chemical species can represent directly the number of correct guesses. However, this signal has to be linear to be easily interpreted, which makes the combining part non-trivial. The general design is shown in Figure 5.2. Because of the symmetry of the Mastermind game, we only have to design the three different parts (evaluation, combination and result) shown in the Figure.

5.1.3 Position evaluation

We choose to produce a species (dubbed *reporter*, see previous Chapter) when the guess is correct and nothing when the guess is incorrect. There are two ways to do so: either use the guess species as an activator for the reporter, with an inhibition if the guess is wrong (see Figure 5.3), or use the bistable to continuously prevent the creation of reporter and inhibit this activity with the guess (double negation, our actual design). The main problem with the first option is the leaky nature of the inhibition process. Even when the wrong species is introduced in the system, some reporter is produced. This leak will then be amplified by the successive layers of signal combination, giving potential false positives.

We use the opposite approach. An autocatalytic template is added to the reporter, with an indirect inhibition from the bistable circuit (Figure 5.4). The intermediate species is shared by

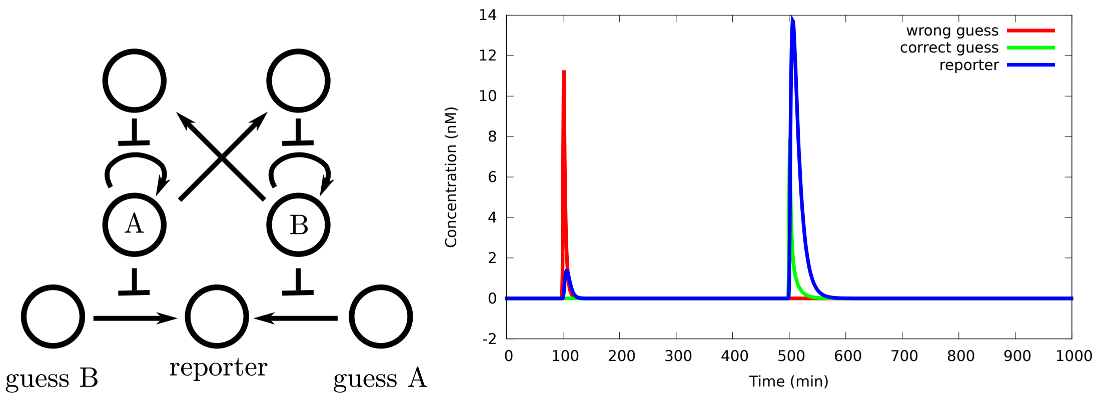


Figure 5.3: Alternative design for the position evaluation (left) and its response to both possible inputs (right). We can see that a low concentration of signal is still created with the wrong input (leak).

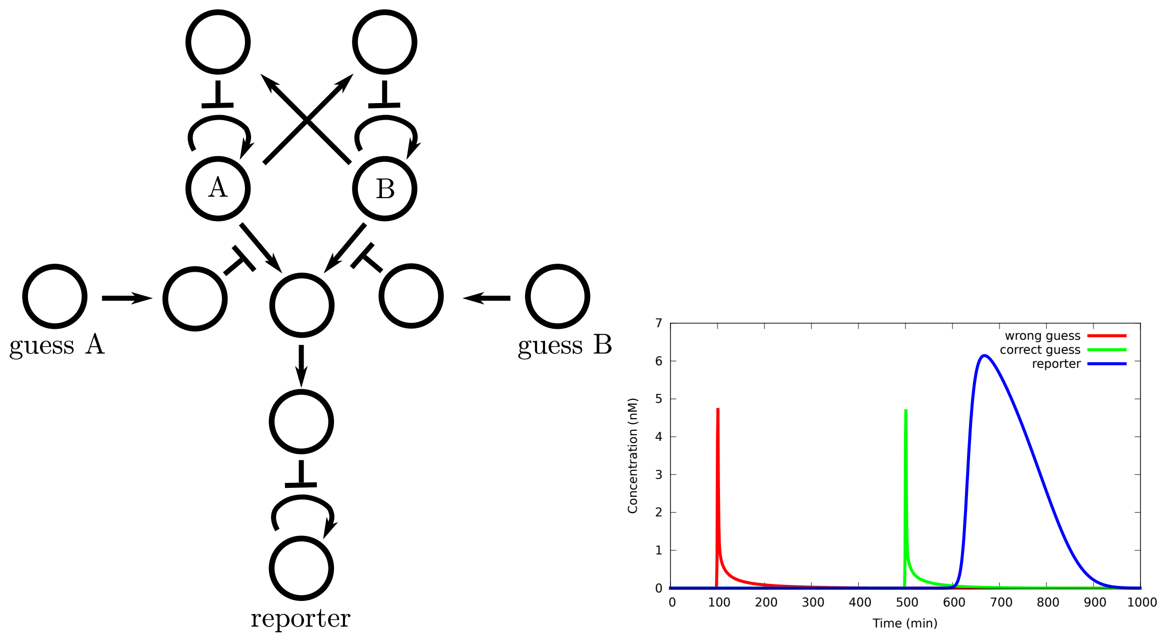


Figure 5.4: Indirect evaluator design. The reporter species can only be produced with the correct guess. Note the delay introduced by taking an indirect path.

each states of the bistable circuit to save sequence space[1]. We now prevented the leaks, but we have instead a sensibility problem. As described in the previous Chapter, we finally used the load effect to sustain the presence of the guess species long enough to get an effect.

We then need to optimize this part of the system to avoid enzymatic saturation later on. Indeed, it is obvious[2] that making copies of such a large system will have major non-linear effects on the enzymes.

Templates concentrations were optimized using CMA-ES [92] until reaching a satisfying response to input. To do so, the reporter circuit was simulated for about 2000 minutes. To get

[1]Remember that each time we add a node, we will have to design a sequence with the corresponding specifications.

[2]in retrospect, at least…

a saturation similar to that of the final system, dummy autocatalytic and activation templates were also included in the system. The reason to choose this technique instead of simply optimizing on the complete system is twofold. First, all evaluators will be identical in the end, so we only need to evolve one. Second, autocatalytic modules and empty templates are fast to simulate, so evaluations by the optimization algorithm are faster.

At 500 and 1500 minutes inputs were added; first the input corresponding to the state of the bistable circuit, then the one corresponding to the other possible state. The long delay was used to ensure that the steady-state is reached at the time of input. The optimization target was then set to have a large response when the correct input is added and no answer otherwise. Once the optimization was done, template concentrations were adjusted so that the system would be symmetrical. We also rounded those concentrations to the nearest integer, to take into account the maximum precision of the experimental material.

5.1.4 Combining results

At this stage, we have again a design choice to make. On the one hand, we can choose to encode the amount of correct answers digitally, for instance by reusing the two-bits counter presented in the previous Chapter. However, it is much easier to simply connect each reporter species together to get an analogue representation of the number of correct guesses. The more correct positions, the larger the generated concentration. This design also requires much less parameter tuning than additional toggle-switches [11].

Combining signals is not as straightforward as could be hoped. Connecting all the reporter to the same species leads to a non-linear response due to the exonuclease. The more this species is produced, the more effort will be needed to increase its total concentration further. The effect is not so obvious for this simple implementation of Mastermind (Figure 5.5), but it mostly means that the design cannot be scaled up. We chose instead to use the layered design presented in the previous Chapter to ensure this scalability. This design is not without flaw, as the balance between different groups becomes non-linear instead, but it was possible to use CMA-ES to automatically solve this problem. Additionally, inhibiting an autocatalyst tend to have more impact than simply producing a final species, as can be seen by comparing the concentration of I_{56} and s_{55} in Figure 4.11.

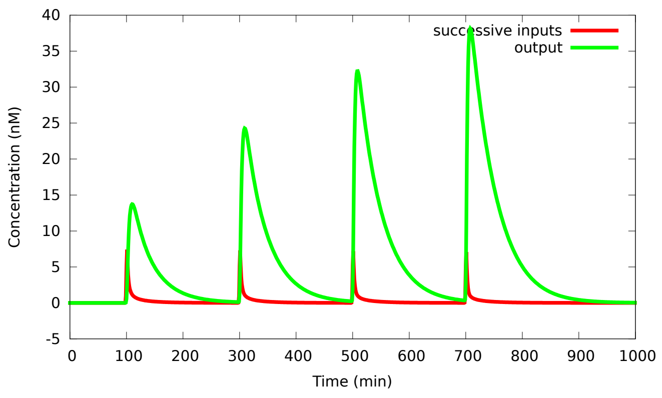


Figure 5.5: Non-linear response to a growing number of correct guesses. The higher the output concentration, the more impact the exonuclease will have on it.

5.1.5 Misplaced guesses

Technically, to implement the full rules of the Mastermind game, we need to add one additional subsystem. As we mentioned in the description of the game, Mastermind also includes a number of misplaced guesses in the hint to the player. This is done by computing how many symbols could be in a correct position if an optimal permutation was applied. This computation excludes the guesses that are already corrects, but such guesses can be easily removed upstream of this system by the position evaluation modules. We note $s' = s'_1 s'_2 \dots s'_k$ the maximum substring of unmatched positions and $g' = g'_1 g'_2 \dots g'_k$ the guesses corresponding to those positions.

Implementation of permutation is not straightforward, as we would get hit by the combinatorial complexity of such system. It might still be manageable for two or three symbols and a few positions, but this design would not scale up. Instead, we can note that we do not care about the actual optimal permutation, as this information is obviously not given to the player. We can instead count how many of each symbols are remaining unmatched, and compare this value to the number of times this specific symbol appears in the incorrect guesses. Formally, we get

$$misplaced = \sum_{a \in \Sigma} \min\left(Card(\{i | a = s'_i\}), Card(\{i | a = g'_i\})\right)$$

We saw how to sum results in the previous section. What is left is to be able to compute a minimum over two values, that is, to be able to perform a comparison. Comparing the concentration of a species a to a species b can be done efficiently in the DNA toolbox by using an inhibited activation template connecting one of the compared species (without loss of generality a) to a reporter. The inhibition is done by other species, here b, and is calibrated so that there is a (noticeable) increase of the reporter if and only if the concentration of a is higher than the concentration of b (comparator in Figure 5.6). Then, it is possible to combine the comparator

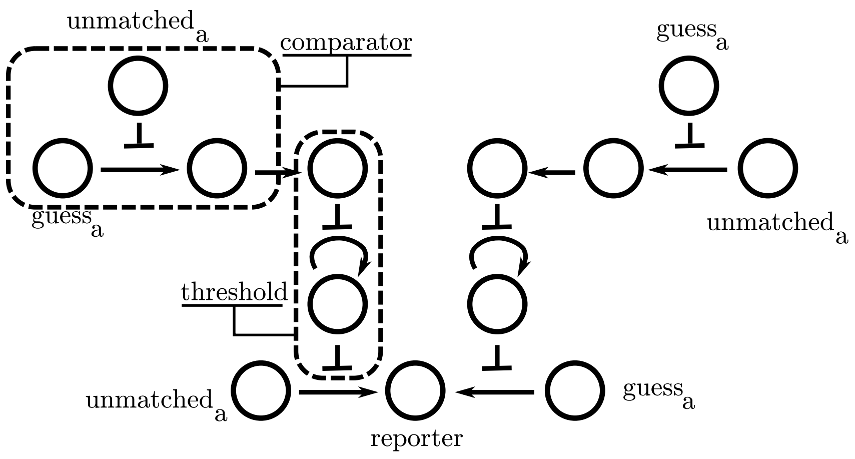


Figure 5.6: Amount of misplaced guesses for a given symbol. The effect of this system is to find the minimum between the number of guesses of symbol a and the number of unmatched symbol a. If the guess is higher, it will pass the comparator check and trigger the threshold, allowing the number of unmatched symbol to activate the reporter. Note that the two inputs (number of guesses for symbol a and number of occurrences of a in the unmatched positions) are simplifications, and represent two specific circuits based on combining specific guesses and status before connecting the results to this network. Note also that the two comparators cannot have exactly the same (mirrored) template concentrations, since in the equality case, one comparator has to be accepting and the other blocking to get the correct result.

to a reporter by using a threshold. This threshold design has been exploited in the position evaluation mechanism: we use an autocatalyst constantly generating an inhibitor (Figure 5.4). This autocatalyst can in turn be inhibited, but if the inhibition is not strong enough to stop its autocatalysis (which represents the threshold), no signal will be observed. Conversely, if the autocatalyst is temporarily stopped, the output signal will be mostly independent of the input. There is still an input dependence around the threshold value, but it is possible to have a clear cut due to the roughly digital nature of our input. The schematic idea of this circuit for a given symbol is shown in Figure 5.6.

We can then note that this system is fairly large and will saturate the enzymes in the full system. Moreover, as we noted in the previous Chapter, giving the number of misplaced guesses is unnecessary in the simple case we are implementing, which is why this system was not included in the design. It displays however interesting patterns that can be created with the DNA toolbox. Additionally, such patterns can be reused in a multitude of systems, and far from constraining the designer, they help him not “reinventing the wheel” every single time. In Chapter 7, we will see how new patterns can be automatically discovered by using an evolutionary algorithm.

5.2 Scripted generation of systems solving the 3-SAT problem

In this section we will discuss how patterns, such as the ones we talked about so far, can be used for the scripted creation of systems. This opens the door to the creation of high-level language compilers [145] which could build efficiently the systems described by combining such patterns. As a proof of concept, we apply this approach to 3-SAT, a classical computer problem [146].

As a quick reminder, in the 3-SAT problem, we are working with boolean formulas such as this one:

$$(x \vee y \vee z) \wedge (\neg x \vee y \vee t)$$

x, y, z, t and so on are called *literals* which can take either the *true* or *false* value. $\vee$ represents the logical "or", $\wedge$ the logical "and". The problem is to find if a given formula can be satisfied, that is, if there is a particular set of values for the various literal that makes the overall formula *true*. This problem is one of the most famous representatives of the NP class, for which there is, in the general case, no better algorithm than trying all possibilities. This property makes the 3-SAT problem a nice benchmark for a given computation paradigm, especially for those based on parallelism. It is then no surprise that DNA computing was also applied to solve this problem [82, 14, 83].

This work appeared as part of Reference [143].

5.2.1 Related work

For molecular programmers, being able to describe systems in a high-level language, as close as possible to natural language, and have it automatically turned into an actual detailed experimental setting is a huge improvement toward larger systems. Similarly to what happened with the advent of the C programming language [147] in computer science, a new class of molecular programming could emerge from this approach.

Many steps have been made in this direction. Luca Cardelli proposed a theoretical DNA strand algebra [148] that could be used as a low-level assembly code for DNA computing. Building on this, methods were created to compile DNA Strand Displacement (DSD) systems [149, 150], which eventually gave the birth to VisualDSD [96]. This piece of software provides a very comprehensive user interface and multiple tools for the design of DSD systems. Qian and Winfree also developed their own program which could compile, by successive intermediary steps, boolean logical circuits into DNA sequences that could then be mixed in a test tube. The resulting systems are impressive [10, 41] and illustrate well the benefits of this approach.

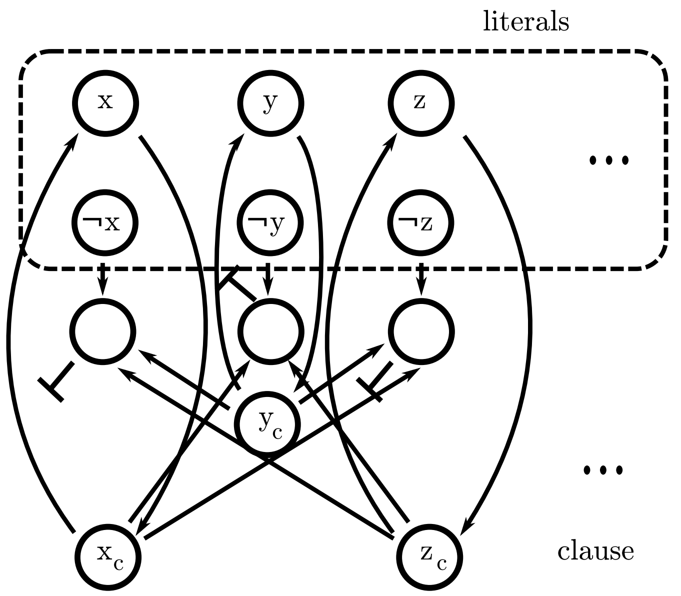


Figure 5.7: The two patterns (literals and clauses) used by the compiler. When the formula is parsed, we keep track of all the literals used. Each literal is represented as a set of two species. The clauses themselves are then encoded with an indirect 3-stable using connections to the relevant shared literal value. Here, the clause encodes $x \lor y \lor z$. Note that a state in a given clause is also inhibited by the literal species corresponding to the opposite truth value. For instance, the species representing the literal $\neg x$ will inhibit all x_c states in the clauses of the system.

5.2.2 Compilation

Like in any compiler, the first step is to define a translation of the high-level language to a lower-level one. If no optimization of the code is expected, this can be seen as a simple rewriting process. The higher-level language is defined by a grammar and each elements are attributed an equivalent at the lower-level. Note that this task can be made much easier if the grammar has some nice properties, such as being unambiguous. Here, the high-level language is simply that of the 3-SAT formulas, which is very rigid and as such can be parsed without problem.

Once the code is parsed, we can write its lower-level equivalent. The compiler uses predefined basic patterns and combine them to form the final system. We already described some of such patterns, like the threshold (Figure 5.6), the comparator or Padirac *et al.*'s bistable circuit [11].

For the problem at hand, we represent each literal by two specific species, one for the value *true* and one for the value *false*. Each clause is represented by an indirect 3-stable (Figure 5.7): the stable state represents which literal give the value *true* to this clause. This design is justified by the fact that only one *true* literal is enough to make the whole clause *true*. We use an indirect autocatalysis to be able to share the value of the literals. At parsing time, we remove duplicated clauses as they do not change the satisfiability of the formula, but make the DNA toolbox system larger.

The system is then implemented in the graph file format of DACCAD (see Annexe B.4) which can also simulate the result. We say the steady-state of our system is *coherent* if and only if no literal is set to be both *true* and *false* (*i.e.* both species present).

Theorem 5.2.1. *An instance of 3-SAT is satisfiable if and only if its steady-state in the DNA toolbox equivalent is coherent.*

Proof. $\Rightarrow$: If the 3-SAT formula is satisfiable, then by definition there is a set of values for the literals that verify this formula. Let us consider the corresponding state. By construction of the indirect 3-stable circuits, this (or a degenerated version[3] of this) is a stable state, so this system has at least one valid stable state. Suppose that the system is in an incoherent state. Without loss of generality, let x be the literal that is in an incoherent state. Then x is used to verify some clauses, while $\neg x$ is used for others. By construction, the concentrations of the x and $\neg x$ species are low, so the state can only be steady if any modification to the clause 3-stable circuits would lead to an other incoherent state. However, we have shown that there is at least one coherent steady-state, so there exists a sequence of thermodynamically favorable modifications in the system that leads to this other state. So the incoherent state is not stable.

$\Leftarrow$: Trivial by construction: the truth value of each literal verify the formula. □

Note that this proof only works for idealized systems. As we showed in the previous Chapter, enzymatic saturation can change the stability of systems. For a very large formula, the number of species might prevent a correct computation if the concentrations are not carefully adjusted. In the following examples saturation was disabled to bypass this issue, but for an actual *in vitro* implementation adjustments are required. This might be mitigated by simplification of very large systems: it has been shown that the probability for a random 3-SAT formula to be satisfiable is linked to the ratio of the number of literals over the number of clauses, with a very clear phase transition[151].

We can note that we get an implementation of a 3-SAT problem in linear size (in term of sequences and templates). However, the systems we are implementing are *frustrated*: many autocatalytic states are inhibiting each others, leading to particular relaxing dynamics taking an exponential amount of time to reach the steady-state [152].

The last step, the actual DNA sequence design, has not been realized yet, but could be completed by interfacing with NUPACK[93], for instance.

[3]if the value of one of the literal does not mater to verify the formula, the concentrations of both its species will go to zero

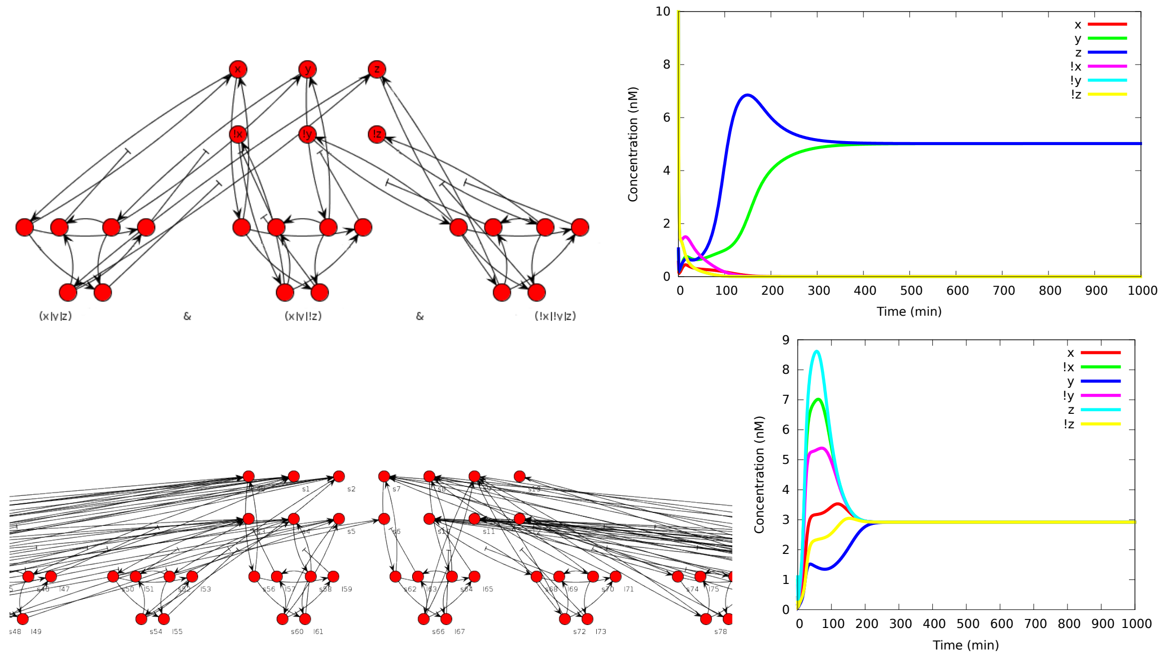


Figure 5.8: Top: Simulation of a simple 3-SAT problem. In this specific case, the system settles on both y and z *true*, with the value of x undecided. Bottom: Simulation of a large problem. In this case, no agreement was possible. Conflicting literals are depicted on the right.

5.2.3 Simulation results

The system created were fairly large, but were correctly simulated, hinting that bigger systems yet can be designed with DACCAD. Figure 5.8 shows the results for $(x \vee y \vee z) \wedge (x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$ and a very large non satisfiable formula (7 literals and 16 clauses).

5.3 Conclusion

In this Chapter, we saw two complementary ways to design systems through DACCAD. The first part showed how DACCAD's user interface can be used to create systems with complex behaviors. It also explained how to solve saturation problems using CMA-ES [92], which is a built-in tool of the program.

The second part showed how to use the graph representation of DNA toolbox systems to compile instances of the 3-SAT problem into a chemical equivalent. This approach can be extended to many problems, provided a library of basic patterns can be set to serve as assembly language. We will see in Chapter 7 how some new patterns could be found and how to deal algorithmically with complex problems.

Finally, we should note that the two approaches presented in this Chapter are in no way mutually exclusives. It is in fact quite possible to use a script to compile some large system before doing minor modifications with the graphical interface.

Chapter 6

Extending the model to debug side reactions

In the previous Chapters, we mostly assumed that the reactions described by our model were perfect: no polymerase error when generating a new strand, no unexpected hybridization and so on. We only reconsidered this assumption in Chapter 3 when the experimental reality flagrantly violated this view, leading us to enrich the delay gate model with imperfect stoppers. Similarly, strange experimental results with DNA toolbox systems are likely due to side reactions not taken into account by the model of Chapter 2, and require a concrete investigation.

In computer science, many tools, such as debuggers, are always available, allowing the step-by-step execution of programs, detailed data analysis and more. This close inspection allows us to determine exactly what the problem is. While more often than not this process is extremely time consuming, as the origin of the bug might be unknown, it is still a convenient method. In DNA computing, however, we do not even have the luxury of this detailed inspection. Due to the lack of interface with the system, we end up with an experimental black box and are reduced to play with its inputs (the DNA molecules we use, the buffer and so on) hoping to get some insights from the (noisy) output we observe. Moreover, some of those outputs may be completely incorrect due to a human mistake, making it even harder to find where the real problem can be. This lack of data may be mitigated by the recent possibilities to test a large amount of different inputs with only one experimental setting [129]; microfluidic mixing devices [86] can be used to get a better control of what is inside the black box; automation of experimental preparations [55] can remove the human factor. Finally, it is also possible to do many replicates of the same experiment and stop the evolution of their respective systems by dropping abruptly the temperature below the activity range of the enzymes. Additional analysis, such as species concentration titration, are then be performed on those samples to get a sense of the precise evolution of the system with time [153, 67].

Once enough data have been collected on a specific “buggy” system, hypotheses are made and then tested, as is the classical approach of scientific research. For instance, if a DNA sequence is suspected to have unwanted secondary structures, a confirmation experiment would use the same system structure, but using a different DNA sequence. In this Chapter, we show how we can extend the DNA toolbox model to take into account such hypotheses and simulate quickly the impact they have on a given system. While such simulation is not enough to conclude whether or not a supposed problem is indeed responsible for the observed behavior, this approach gives us a way to orient the search and plan meaningful experiments. In particular, if the simulation fails to reproduce the observation, it means that the hypothesis is wrong or incomplete.

In the first section, we describe some basic problems that may arise when using the DNA toolbox. While those problems have been mostly removed from the standard implementation of the DNA toolbox, any change to the framework (such as using a different polymerase) requires to give them careful consideration. Then, we try to estimate what assumption we made in the model (see Chapter 2) might be hard to satisfy, and what problem would then arise. In the third section, we present the methods we use to detect potential errors in faulty systems. Finally, we apply this method to explain some unexpected experimental results.

6.1 Problems already taken into account in the DNA toolbox

Montagne *et al.* [67] and, later, Padirac *et al.* [43] took steps to prevent unwanted reactions between elements of the DNA toolbox. Namely, DNA strands used as template are chemically modified to ensure the following properties:

- Modification of the template $5'$ end to prevent degradation by the exonuclease. However, this modification does not prevent the enzyme from attaching to the template, forcing us to take the concentration of free template into account to compute the exonuclease saturation (see Chapters 2 and 4).
- Modification of the template $3'$ end to prevent extension by the polymerase.
- In the nicking enzyme recognition site of the output, a DNA nucleotide has been replaced by an RNA one (U, uracil). This modification does not affect the action of the polymerase,

but prevents the nicking enzyme from attaching. This lowers the saturation effect of templates on the nicking enzyme.

Signal sequences cannot be modified, as they are generated on the spot by the polymerase. However, their sequences should be designed to avoid the following problems:

- Multiple G (guanine) in a row. While we only consider Watson-Crick base-pairing [24] in our model, DNA can congregate in much more complex ways. In particular, DNA with four Gs in a row can fold into a structure called G-quadruplex. In the context of the DNA toolbox, this structure will prevent the DNA strand from interacting with its complementary and thus should be avoided.
- Multiple C (cytosine) in a row. Since signal and templates are complementary, multiple C in one means multiple G in the other. See the previous item.
- Multiple nicking recognition sites. Having more than one would allow the nickase to cut the sequence in the wrong position, creating unwanted products. Note that this extraneous site may overlap between two sequences (Figure 6.1, b.).
- Complementary of the nicking recognition site. This will cut the template, effectively changing the system. Depending on where the template is cut, it may becomes a sort of signal protected against degradation. The resulting behavior can be hard to predict.
- Self-hybridization. A signal species should be prevented to be able to interact with itself (or any other species). There are multiple possibilities, depending on which part of the sequence gets hybridized (Figure 6.1, c. and d.). This will at least change the rate at which the species attaches to templates and at worst allow the creation of new species or pseudo-templates.
- Similarity (first order leak). If the sequence of two signal species share too many nucleotides, especially at the $3'$ end, those species may be able to activate each other's templates. This will create unexpected connections in the system, changing its behavior.

They also optimized both buffer and reaction conditions. Buffers have specific salt concentrations and pH, as well as an addition of multiple molecules for enzymatic stability such as trehalose, DTT (Dithiothreitol) and BSA (Bovine Serum Albumin). dNTP concentrations were adjusted to allow long term operations in a closed environment while keeping phenomenons such as zeroth order leak as low as possible. Reaction conditions (enzymes, working temperature)

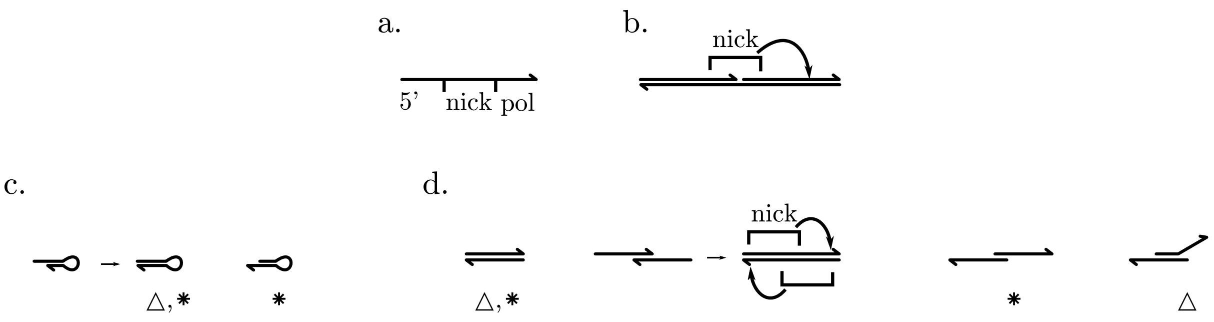


Figure 6.1: Possible problems with interactions. **a.** Three areas of a signal strand. The effects of a copying mistake from the polymerase are different depending on the area. Overall, any modification will decrease the affinity of the sequence for its template. An error in the nicking recognition site (nick) will disable the nickase activity. An error at the $3'$ end (pol) will possibly prevent the extension by polymerase on its template. **b.** Effect of having the sequence recognized by the nickase shared on a template. An erroneous cut will happen, producing incorrect species. **c.** Possible self-hybridization. Some will protect the strand against degradation ($*$) or nicking ($\triangle$). **d.** Possible hybridization of two signal strands. Some conformation may be protected against degradation ($*$) or nicking ($\triangle$).

were also taken into consideration. The DNA toolbox was thus optimized to run systems for as long and as stably as possible in a closed reactor. A complete description of those experimental conditions and design guidelines can be found in Baccouche *et al*'s Method [101].

However, despite all those precautions, we still observe the emergence of what is informally called "the divergence". In EXPAR [154], a system similar to the DNA toolbox, it was noted that such behavior is due to the appearance of autocatalytic parasites which hijack the enzymatic machinery [155].

6.2 Relaxing the model limitations

Since this divergence is not due to the DNA sequences (or at least not in a way we know of), we have to examine in more details the hypothesis we made in Chapter 2. Namely, we cannot assume that the actions of enzymes are perfect or without intermediary steps anymore. This gives us a pretty comprehensive list of what can (and probably do) go wrong in an *in vitro* implementation. We can then enrich the model with part or all of those additional reactions representing enzymatic errors.

- Polymerase: Bst Large Fragment, the polymerase used in the standard implementation of the DNA toolbox, lacks any proofreading ability [156]. This means that it will make mistakes when copying a template, using spuriously the wrong nucleotide in some places. Those mistakes will have different effects based on where they are located (Figure 6.1, a.). The polymerase can also extend a DNA strand past the end of the template, adding a few more nucleotides. In most cases, those additional nucleotide will only prevent the new signal strand from being compatible with its target templates. There are however a few chances that the new strand will be able to attach its $3'$ end to itself or other signal strands, causing the problems noted in the previous section (Figure 6.1, c.).

- Nickase: this enzyme might cut at the wrong position, creating shorter or longer DNA strands than expected. They should, however, behave almost normally: a longer strand will still attach on a template at the correct position, a shorter one will only be less stable. Nickase might also attach to the wrong structure (like a template duplexed with an input signal strand, noted $temp_{in}$ in Chapter 2), making it too stable, preventing extension by the polymerase and saturating the nickase. Finally, the nickase may be too slow to cut an extended strand. In this case, the two-domain strand might be able to denature from the template and act as a sort of new template. This is mitigated by the reverse reaction, where it attaches back correctly to a compatible template[1] and is cut afterward.

- Exonuclease: of the three enzymes, this is the one that has the least chances from creating a parasitic species. Even if a DNA strand is only partially degraded, it will not be able to react in any new (or meaningful) way. The exonuclease may also accidentally destroys templates, but this will only impact the global reaction speeds of the system. Overall, this enzyme will instead digest new species that were created by accident and delay the apparition of such parasites, acting as an error-correction mechanism.

All those mechanisms are fairly unlikely. The main reason why divergence still happen is likely due to the sheer number of molecules reacting together, combined with the emergence of an aggressive autocatalytic behavior and/or mechanisms preventing the degradation of the parasite by the exonuclease. Indeed, the standard model predict that, under normal conditions, even legitimate autocatalytic templates will not be able to sustain the production of signal if their concentration is too low. Even taking into account exonuclease saturation, problematic DNA species are created one at a time, which is orders of magnitude below the theoretical requirement.

[1]Considering the number of molecules in the solution, it is unlikely that this will be the exact same template it separated from.

6.3 Debugging side reactions: method

Our strategy to debug time traces of experiments relies on taking into account various possible faulty operations using a stochastic algorithm, and check the simulated behaviors match reality. Then, we can check what reaction paths generated the incorrect trace. The debugging takes place in the following four steps.

First, check that all DNA sequences used in the system verify the DNA toolbox guidelines [43, 101]. This step should only be a routine check if the sequence design was done following the specifications of Padirac *et al.* [43]. It might also be useful to check the troubleshooting Section of Baccouche *et al.* [101]. Any problem at this stage should be corrected as much as possible, as it will make the analyze of other potential problems harder.

Second, based on the observation, we choose to add some reactions that are most likely to cause the faulty result. For instance, a progressive signal reduction may be a sign that the exonuclease is degrading templates; the divergence is a sign of strong parasitic species created by polymerase errors. Allowing into the model all side reactions from the previous Section is a costly possibility, but should be avoided due to the combinatorial state explosion: each of this side reactions will introduce new classes of incorrect DNA species, pushing against the limits of what can be simulated.

We can then move on to simulating the system using the Gillespie algorithm [157]. This is a stochastic algorithm, as opposed to the predictive one we have been using so far. Indeed, to avoid the combinatorial explosion of species, we try to simulate a limited number of DNA strands. At each simulation step, we select among all available reactions which one will happen next. This selection is weighted by the kinetics of those reactions: a fast reaction will have more chances to occur than a slower one. The state of the system as well as the list of possible reactions are updated: a new species can allow additional reaction paths, while a destroyed species may close some. Since kinetics for side reactions are not available, we use "reasonable" values, as described in Chapter 3.

Finally, we compare the simulated behavior to the real one. We can quickly iterate among various conditions until a promising match is found. All that is left is to look at the generated reaction paths and analyze their behaviors, do statistics, and so on. Sets of experiments are then designed to check those hypothesis.

6.4 Application to a simple autocatalytic reaction

We focus in this Section on analyzing the strange behavior of an autocatalytic module in an environment without exonuclease. As mentioned before, the exonuclease has a correcting role in DNA toolbox systems. As such, we can expect parasitic DNA species to be much more likely to appear when the concentration of this enzyme is low or null.

Looking at the experimental results on a ramp of concentration of polymerase (kindly provided by Dr. Anton Zadorin and reproduced with his permission, Figure 6.3), we can see a first plateau, then a shallow increase of fluorescence followed by an exponential growth until (presumably) all dNTPs in the solution are consumed.

Based on the standard model, the first plateau corresponds to the saturation of the autocatalytic template. Once the concentration of signal strand in the solution is high enough, the template can be considered to be continually double-stranded: as soon as a signal strand detaches, another takes its place. Most of the fluorescence from EvaGreen comes from double-stranded DNA, so even though more signal strands keep being produced, no significant increase in fluorescence should happen anymore. The best explanation for the next brutal increase is then to assume the apparition of another autocatalytic reaction. Our objective will thus to look for reaction paths allowing autocatalysis.

Sequence check The first step was thus to analyze the autocatalytic sequence. We found that the DNA sequence of the autocatalytic signal allowed two signal strands to hybridize, forming a dimer. Moreover, this structure was blocking the nicking site of one of the two strands. While, in many cases, this would be removed by strand displacement due to the template (that is, the template attaches completely to one strand, freeing the other), there is a chance that the signal strand, elongated by the polymerase, would detach before nicking was available (Figure 6.2). Those chances are even increased by the smaller length of the template-signal duplex, equivalent to that of an inhibitor, making the denaturation a fast reaction. We are then left with a two-domain "signal" strand. This strand can be elongated even more, as long as its last nicking site is disabled while it is attached to a template.

Additional reactions Since data with different sequences for this experimental setting are not available, we chose to model this potential deactivation. We add two possible reactions: two simple signal strands can hybridize to give a signal strand with disabled nicking activity, and the reverse reaction, a duplexed signal denatures, releasing the two original strands. Those

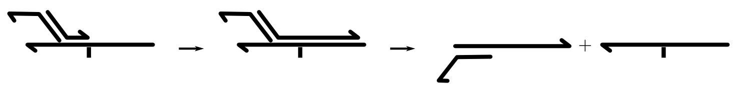


Figure 6.2: Accidental generation of two-domain signal strand. Since the nicking recognition site is not attached to the template, the signal strand is not cut before being released. Not that in most cases, the second signal strand will be removed by strand-displacement, making this outcome unlikely.

reactions can adapt to signal strands of any size, as long as there is an available nicking site.

We add in the model the possibility for new templates to be generated by zeroth order leak from the polymerase: even without primer, the polymerase can attach to the single-stranded two-domain signal strand and generate a new template from it. Note that this process will completely double-strand the signal strand, allowing the nickase to finally cut it. Those new template strands are not modified at the $3'$ end, meaning they can react with polydomain signal strands to get elongated. In doing so, they may end up double-stranding some nicking site on the polydomain signal strand, leading to cutting it into smaller, less stable pieces. The elongated template is then freed and able to interact with the rest of the system.

Stochastic simulation We can then simulate the system. Polymerase error is not taken into account, considering that blocking the nicking site occurs most of the time due to the unfortunate tendency of the signal strands at hand to hybridize to each other at this position. A typical simulation result is shown in Figure 6.4. Note that we are not simulating the dNTP concentrations, which is why the curve is not capped. While we do observe a plateau followed by another fluorescence increase, there is no exponential growth. This primes the suspicion that polymerase copy errors might definitely play a role in triggering this additional exponential growth.

Estimation of the system's real behavior For the reason given in the previous paragraph, we need to add to our model the possibility for the polymerase to make a mistake when copying the nicking site, generating a strand with a faulty nicking site. Other copy errors do not seem relevant to the problem at hand, and can probably be ignored. Eventually, a generated template with a mismatch in (one of) its output domain(s)[2] will generate new signal strands which can still interact with normal templates while being unable to be cut by the nickase. A quick search of reaction paths shows that this can lead to an exponential increase of templates (Figure 6.5). Note that in any case, the ever increasing length of DNA templates will worsen

[2]There might be more than one if the template was generated from a long signal strand, or was elongated afterward.

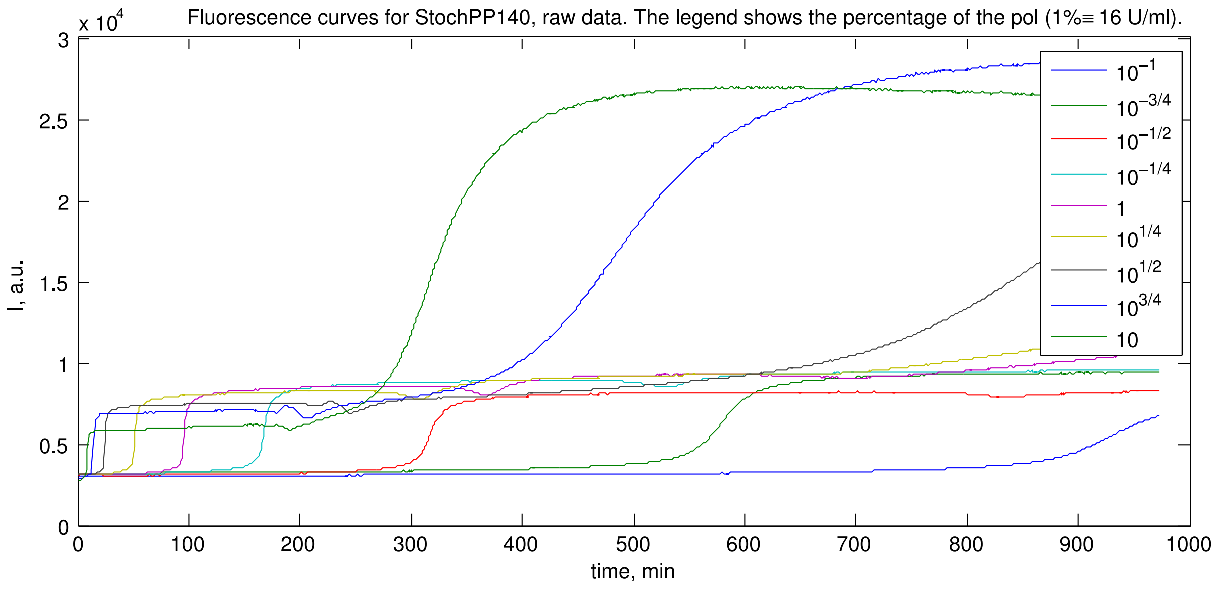


Figure 6.3: Data provided by Dr. Anton Zadorin. Autocatalytic template with no exonuclease. Ramp of polymerase concentration.

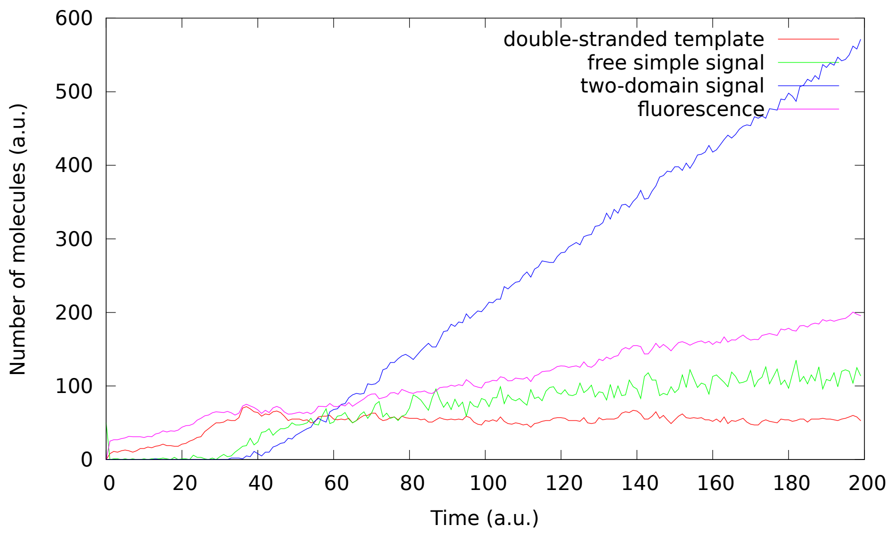


Figure 6.4: Stochastic simulation of the system with the possibility to disable nickase recognition site. Values for reaction rates are similar to those in the simulation of Chapter 3 (slow, average, fast). After reaching a plateau, the system starts creating more templates, which leads to a new increase of fluorescence.

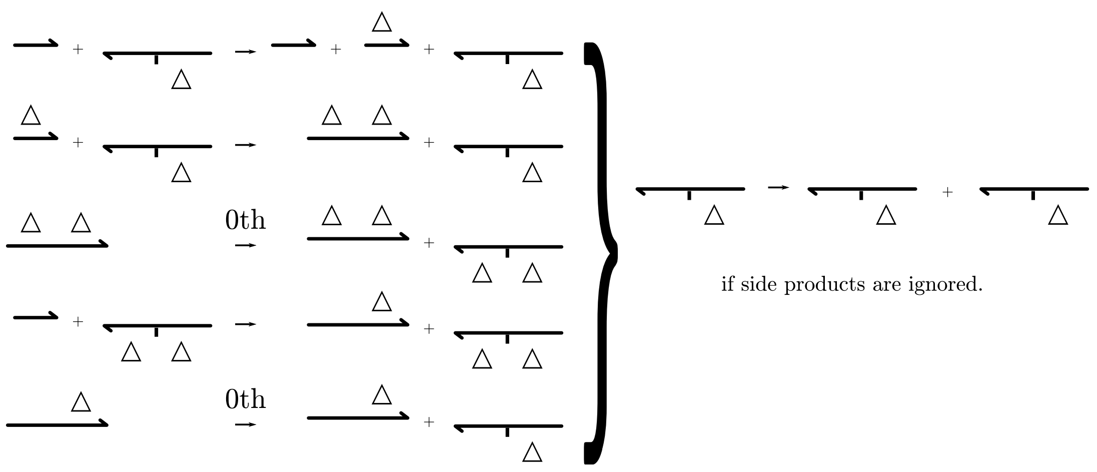


Figure 6.5: Example of autocatalytic path due to a template with a faulty nicking recognition site on its output. Reactions are actually multi-steps, similar to those of the standard DNA toolbox. The triangle (△) mark represents faulty nicking sites. 0th indicates a zeroth order polymerase activity [103]. Note that this cascade produces another species with such autocatalytic path (the template with both deactivated nicking site).

the saturation of nickase, making faulty dissociation (dissociation of a two-domain signal strand before it is cut) more frequent.

6.5 Related works

High-level models cannot capture everything and unexpected experimental behaviors can be explained by leaky reactions missing in our standard model. By using instead a bottom-up approach like in oxDNA [53], it is possible to maintain specific properties of DNA (for instance, coaxial stacking of DNA nucleotides) that may have a huge, albeit indirect, impact on DNA reactions and would accidentally be abstracted by a top-down model. This approach was used, for instance, to optimize the behavior of a strand-displacement based walker [158]. The drawback is of course that modeling more details increases the computation time and/or limits the size of systems that can be modeled. Enzymes add a lot of complexity themselves, as each require many additional operations to be taken into account.

Additionally, the way we chose to simulate additional operations in this Chapter is close to the system from Kawamata *et al.* [159]. Like in their work, we are limited by the exponential growth of the number of species. The way they chose to bypass this problem is to select which species, or class of species, will be most relevant, and limit the computation to those. They successfully applied this latter method to model gene silencing by RNAi [160].

6.6 Toward automatic model generation

By adding new reactions, corresponding to a local disabling of the nicking recognition site, we were able to give insights about the divergence mechanism in a particular wet-lab implementation of an autocatalytic module. However, the simulation shows that this assumption is not enough to explain the observed parasitic behavior in its entirety. In particular, we suppose that it is necessary to add polymerase copying error to the system to capture this parasitic process.

However, instead of going back and try to add new potential reactions by hand until a better fit is found, it could be possible to automate this process. For instance, in some cases, it is possible to infer which reaction are happening from the data. Chattopadhyay *et al.* defined a mathematical method to estimate a reaction network from the variations of species concentrations over time [161]. This methods has the drawback of requiring that a steady-state is reached, and that the species are known and monitored, but is still applicable in a large variety of cases. Alternatively, it is possible to generate sets of reactions without any assumption on which species are interacting, or even on the number of species present, by using a genetic algorithm. A similar approach was successfully implemented by Schmidt and Lipson to automatically find laws describing experimental data [162]. Just using basic knowledge about equation generation from reaction sets, such as those we used throughout this thesis, it would be possible to find efficiently what might have gone wrong. This kind of approach is also similar to a process called counter-example guided abstraction refinement in software verification, where an abstract model is progressively refined based on inconsistencies [163].

The loop would then be completed by doing additional experiments, trying to verify those hypothesis. If those experiments are also automatically designed and combined with robotic assistance, we might get a full automation of science[164], applied to DNA computing.

Chapter 7

Evolutionary optimization of DNA toolbox systems

Up to now, we presented tools that allow designers to quickly test and improve explicit systems. However, more often than not, it is hard to come up with a solution to realistic problems. One workaround is thus to let the computer evolve such systems from scratch. In this Chapter, we describe an algorithm which generates automatically DNA toolbox systems verifying some specified characteristics (in our case, producing particular dynamics or behaviors). Those characteristics are encoded as a score (or fitness) that will be attributed to individual instances generated to solve the problem. We also show that those characteristics can be virtually as general as the user wants, with the caveat that the fitness gets harder to define. We also show that we can learn design tricks from the search process itself, not only from the results, and reuse the patterns that are found in other contexts (see for instance Chapter 5 whose comparator and threshold mechanisms were first observed through evolution).

The work presented in this Chapter was presented at the 13th ECAL conference [141]. Note that an improved version of BioNEAT, ERNe has been submitted by Dinh *et al.* [140].

7.1 Introduction

The game of rock-paper-scissors, while being simple, can actually lead to interesting dynamics when it is played multiple times in a row. In particular, each player will try to "read" their opponents in the hope of getting the upper hand. However, if psychological factors are not taken into account, that is, if players are purely logical, game theory predicts that after a while, the optimal strategy becomes to play randomly with no bias among the three possible moves

[165]. Variations of the basic rules exist, but are expected to display the same kind of behaviors (from the point of view of game theory) as the classic three moves.

Interestingly, this game can be a good description of many mechanisms ranging from reproductive strategies of some species of lizards [166] or bacteria [167] to oscillations in a gene regulatory circuit [168]. In all cases, there are three possible moves, each strong against another and weak against the remaining one. This usually leads to dynamical behaviors where the different players are constantly invading each other, forming complex spiral structures in two dimensional systems[167, 169]. Even real life examples, such as the lizard example, display oscillations in population size, with a turnover of approximately six years, based on the field data of [166]. Those dynamics may degenerate into a uniform population depending on the initial conditions, or such parameters as the mobility of the players. On the other hand, they may also occur even in a well-mixed system, where there is no spatial compartmentalization to protect diversity, if a given move gets stronger when it is less frequent [170] or if the system never stalls, like in the repressilator [168].

However, all those examples either suppose or require that a given individual will always "play" the same move. Indeed, the lizard will always have the same size and coloration, bacteria the same genotype and genes in the repressilator are not expected to arbitrarily change which target genes they inhibit. From a strategic point of view, more possibilities open when each agent can decide, at each time, which move he wants to put forward. In such a case, some form of knowledge of the opponent becomes necessary in order to infer his probable next move and play accordingly. This knowledge is obtained from two sources: cheating and analysis of the opponent previous moves. "Cheating" here designates the fact of obtaining clues about an opponent from its behavior just prior to the game, not in the negative sense of making a game uninteresting by bypassing the rules. Note that cheating in this sense is both an integral part of most human plays and of biological strategies, and in any way is an essential ingredient of any physically instantiated game. In fact, instantaneous moves and decisions are not possible in a physical world, which means that information is always leaked somehow. This fact was used by the Ishikawa laboratory in Japan to program a robot hand [171] reacting fast enough to hand gestures to be able to always win against a human[1].

While both cheating and strategic analysis requires significant abilities and are generally associated with intelligent players (or at least, players with intents), we wanted to demonstrate in this work that purely molecular systems are also capable of intricate strategies, whose complexity

[1]Video online at http://www.k2.t.u-tokyo.ac.jp/fusion/Janken/index-e.html

can be comparable to that of real players. Moreover, those strategies are simple enough to hope for an actual *in vitro* implementation, demonstrating that the Turing universality of DNA computing systems [16, 6, 148] is not only theoretical.

The individuals we evolved were defined as entities from the DNA toolbox [67], relying largely on the straightforward graph representation of those systems.

Individuals were evolved through an adapted version of NeuroEvolution of Augmenting Topologies (NEAT) [172], dubbed bioNEAT, using a fitness function based on how well they fared in a population-wide tournament. To our surprise, the apparition of a basic memory was not hard, but was almost immediately discarded, as it was not able to compete against cheating. Due to the necessity of having both players in the same well-mixed environment, it was much more efficient for an individual to actually develop a way to monitor the actions of its opponent while hiding its own move. When pushed to the extreme, this strategy produced interesting dynamics where individuals went through multiple moves before the end of the countdown, trying to settle into a winning position, eventually leading to some fashion of oscillatory systems. The mechanisms used for those purpose were interesting in themselves, including concentration comparators or system with multiple levels of activation, giving, through motif mining, insight into the possibilities of the DNA-toolbox. This showed that indeed, the behavior of purely molecular systems, corresponding to a realistic, directly implementable chemistry, can be interpreted in terms of complex strategic planning.

7.2 Related Work and Current Contributions

Our work builds on multiple sources since it mixes design by genetic algorithm with molecular programming. Game theory was also an important source of inspiration, and was useful to check that our evolved individuals are playing in a way that differs from hypothetical “perfect” players.

7.2.1 Rock-paper-scissors

There are also many previous works related to the game of rock-paper-scissors. However, to the best of our knowledge, they either use individuals which are only capable of playing one move, or link existing dynamics to an instance of the game. The evolution game theory study in [165] is the closest to our work, but lacks the added dimension that comes with dealing with cheating or leak of information [173]. While DNA-based systems can hardly be described as having any form of intelligence, it is easy to rationalize their behavior as cheating, a very real possibilities among human players that is not taken into account in [165].

7.2.2 Motif Mining

The idea of using DNA computing to play games has been previously introduced [174]. Finding systems able to play a game is in itself a challenge that leads to developing new structures, and potentially solve issues related to real life problems. However, the use of evolutionary algorithms [175] stands as a promising candidate to search for interesting reaction circuits. From the structural point of view, the analysis of the fittest individuals of specific runs revealed common functional motifs, which may help build new systems. This is the fundamental approach of synthetic biology, in which biological modules are recombined to perform engineered operations [176]. In particular, it was interesting to note that, although actual patterns may vary from individuals to individuals, it was possible to classify them into rough generic categories. This could be used to create minimal libraries of structures for dynamic systems, that is, off-the-shelves building blocks like those defined in [177]. Such libraries would in turn allow the fast and reliable development of complex DNA-based systems. While, in our case, the structures evolved by the algorithm are possibly not generic enough to be useful in any given context, they still have potential applications for the design of a variety of such systems.

7.3 Model

7.3.1 The DNA toolbox and BioNEAT

One interest of the toolbox in the scope of genetic algorithms is that any modification of the "genome" of an individual (that is, the sequences and templates it is made of, not to be confused with the hypothetical genome their actual DNA strings are encoding) still yields a valid individual (albeit a possibly uninteresting one), and that a wide range of possible behaviors are very few modifications apart. For instance, bioNEAT (see next Section) can jump in two steps from the Oligator [67] to Padirac *et al.*'s bistable system [11], as shown in Figure 7.1. This helps the algorithm navigating the search space more efficiently, as well as preventing, to some degree, the trap of local optima.

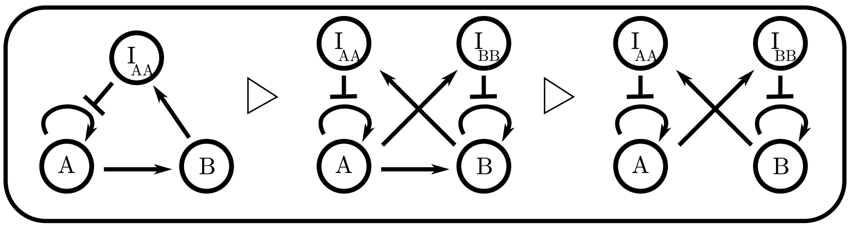


Figure 7.1: Graphical representation of systems from the DNA toolbox. Nodes represent sequences while arrows represent templates. The Oligator (left) can be mutated into a bistable in two steps. First, an autocatalysis connection B to B with an inhibition from A is added. Then, the activation from A to B is removed. Note that those two operations may happen in any order.

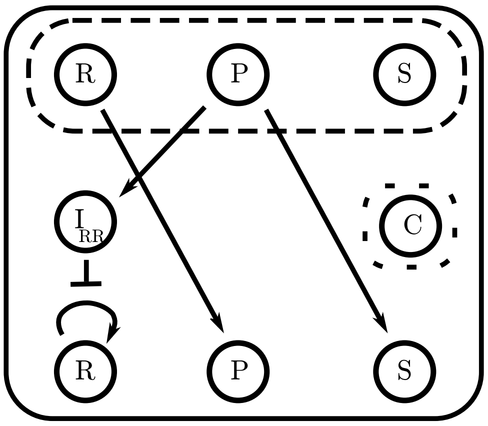


Figure 7.2: Simple cheating individual displaying both direct and indirect monitoring. Nodes in the dashed box are references to the opponent's sequence (up) or to the clock (right). By default, this individual will play rock (R). If its opponent plays rock or paper (P), it will update to play the winning move. Note that this individual does not use the clock.

7.3.2 Individuals and encoding

The individuals we consider are chemical reaction networks playing rock-paper-scissors. Each possible move (rock, paper or scissors) is mapped to a specific chemical species (DNA sequences, more specifically signal sequences from the DNA toolbox). Those species are fixed in advance, so that they are always present. Individuals also have references linking to potential opponents' corresponding sequences. The main goal of this interface is to allow individuals to react to the opponent's moves and adapt their strategy over time. Finally, all individuals have a reference to a common clock species, giving them a sense of time. An example of individual is shown in Figure 7.2.

Individuals are pitted against each other in matches made of ten rounds. The beginning of a round is marked by a spike from the clock sequence. At the end of a round, roughly 20 times the clock's half-life later, an individual's move is decided by which of its move sequences has the highest concentration. If the two highest or all such concentration are not different by at least a given threshold, the move is considered invalid, granting the victory to the opponent. Individuals can potentially memorize their opponent's strategy, since there is no reset between rounds.

7.3.3 Simulations

The simulation itself was kept simple, with a model similar to that of Padirac *et al.* [11]. In particular, this model is slightly different from the one described in Chapter 2 in that it does not take into account enzyme saturation. This prevents some advanced strategies (since saturating enzymes may be in itself a way to kill one's opponent, thus winning by default) and allows individuals to virtually grow without limitations, continuously increasing their size. Since enzymatic saturation creates hidden couplings between the nodes [89], removing it was taken as a step to insure the readability of the results. Thanks to this, the behavior of the network - and hence the individual's strategy - is directly encoded by the networks of cross regulations between the nodes, and not by various type of competitive inhibitions acting at a global level. Using this simplified model is also a compromise between computational requirements and precision, but, based on Padirac *et al.*'s experimental results, any observed behavior should be obtainable in real *in-vitro* experiments.

7.4 bioNEAT: NEAT for Reaction Networks

The evolution of individuals was done by using a modified version of NeuroEvolution of Augmenting Topologies (NEAT) [172], adapted to perform with simulated individual networks built using the DNA toolbox paradigm instead of artificial neural networks. The evolution itself was performed through multiple runs and tweaking of the fitness function.

7.4.1 NEAT

NEAT is a state-of-the-art evolutionary algorithm designed to evolve both the topology and the parameters of neural networks, while keeping them as simple as possible. This is done by starting from very simple individuals, and progressively complexifying them in a competitive process. This is performed through the addition of new nodes and connections, while at the same time modifying the weight of existing ones.

The major strength of NEAT is that it keeps track of when specific connections or node where added in the ancestry line. This allows to perform meaningful cross-over: identical elements present in two individuals, are automatically recognized and matched during the creation of a new individual from two parents. Additionally, mismatching elements from the fittest individual are also passed along.

NEAT also performs speciation to protect innovation that could require more than one step to find a new, better solution to the problem at hand. Specifically, the size of a species depends on the average fitness of its individuals, preventing one type of solution to completely invade the population. Moreover, speciation is easily performed, since the history of evolution of individuals is saved, giving a straightforward distance between individuals based on the genes they possess.

7.4.2 bioNEAT

Due to the initial ressemblance between reaction network and artificial neural network, NEAT stands as a relevant option for optimizating DNA toobox-based systems. In particular, systems from the DNA toolbox have a straightforward edge/node graph representation similar to neural networks: DNA sequences can be directly mapped to nodes, and connections with positive weights are equivalent to activation links. However, the DNA toolbox cannot be directly implemented using the original NEAT for two reasons. Firstly, additional parameters regarding sequences stability and initial concentration must be added. Secondly, negative links targetting nodes must be replaced by inhibitory links targetting arcs. To address these issues, we introduce bioNEAT, a NEAT-derivative that is able to optimize reaction networks.

A first feature of bioNEAT is to allow the algorithm to not only modify the "weight" of connections (that is, the concentration of DNA template, in our representation), but also the relevant biological parameters (such as the thermodynamical stability of DNA sequences and their initial concentrations). The thermodynamical parameters of the move sequences was fixed to prevent individuals to use extremely stable sequences to saturate the monitoring of their opponents. In the particular case of the experiments described hereafter, we also prevented activations toward the opponent or the clock.

The second feature of bioNEAT addresses the asymmetry between activation and inhibition process that is inherent to the DNA toolbox, and which cannot be modelled as classic neural network links with positive and negative weights. While the sign of a neural weight simply encodes the type of the connection and target a node, a DNA toolbox' inhibitor targets an *edge* (and impact only one of the output from the source node) rather than a node. Moreover, an inhibitor cannot be instantiated without the template it inhibits. As a consequence, bioNEAT

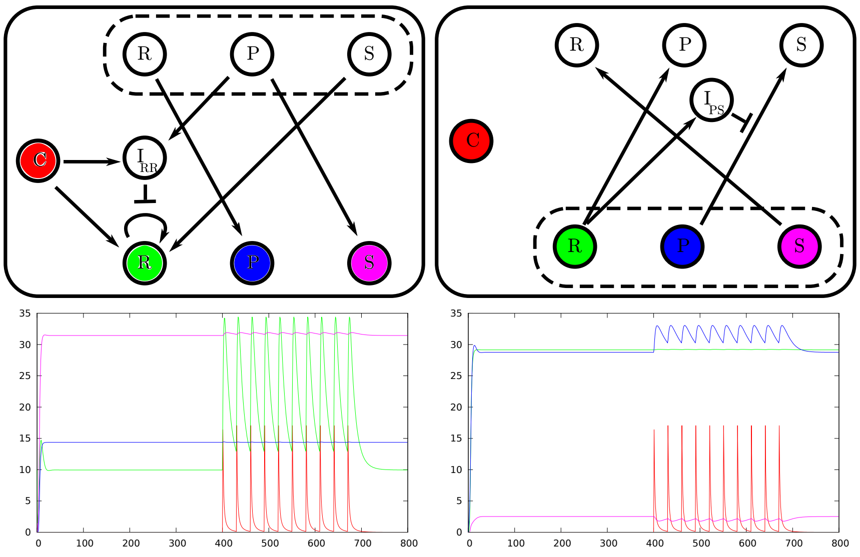


Figure 7.3: Two fighting individuals. References to the opponent's nodes are shown in the dashed box. Top: the actual network of those individuals. Bottom: the corresponding behavior over time. The color code for sequences concentration is red for the clock, green for rock, blue for paper and purple for scissors. The individual on the right has a better comparison mechanism than the individual on the left, as shown by the fact that it has the correct move before the match starts. However, the individual on the left uses the clock to fake switching his move from scissors to rock, which coerce its opponent to update its move to paper. Just before the round is validated, the individual on the left changes its move again to scissors, winning each hands.

protects the addition of an inhibitory connection (and removal of a particular template) during evolution. Then, bioNEAT produces reaction network with inhibitory connections from node to link.

7.4.3 Fitness Score

Scoring of an individual uses a lexicographic fitness function taking place in two steps. First, the individual has to beat the three most basic possible players, playing respectively only rock, paper or scissors. This ensures that our individuals are able to play all moves, and to play them discerningly. Individuals unable to pass this test are awarded a very small fitness, based on the number of rounds they have won, directing the evolution toward basic strategies. On the other hand, individuals which were able to pass the test are awarded the right to enter the second phase.

The second phase is a simple tournament among all remaining individuals: each of them has to fight each of the others. The fitness is then based on the number of correct moves made in total. A sample match is shown in Figure 7.3. Because of this, the evolutionary pressure forces the individuals into an arms race, to be able to defeat as many opponents as possible.

General parameters	
Population size	100
Number of generations	200
Speciation parameters	
Targeted number of species	10
NEAT compatibility parameters	$c_1 = c_2 = 1; c_3 = 0$
Initial speciation threshold	0.6
Minimal threshold	0.1
Threshold update ϵ	0.03
Mutation parameters	
P(Mutation only)	0.25
P(Parameter mutation)	0.9
Otherwise P(Add node)	0.2
Otherwise P(Add activation)	0.2
Else add inhibition	
P(Connection disabling)	0.1
P(Gene mutation (for each node))	0.8
Crossover parameters	
P(Interspecies crossover)	0.01
P(Re-enabling gene)	0.25

Table 7.1: Parameters used to evolve individuals

7.5 Results

Results were obtained by evolving individuals in 10 separate runs, always starting from a uniform population of individuals with autocatalysis on the rock sequence (thus playing always rock). A typical run involved 200 generations of a population of 100 individuals. bioNEAT speciation control loop is adjusted to keep the number of species as close to 10 as possible. Other relevant parameters are shown in Table 7.1. Over the course of the experiment, various kind of strategies emerged before getting outdated or integrated into more complex control systems. However, in our runs, a stable group of species typically appeared after 50 to 100 generations and quickly took over the population until the end of the run. They represent individuals which had developed part or all of the mechanisms explained later in this Section, and the apparent stability was only due to a constant arms race, where individuals kept adding more and more modules, while those who couldn't keep up where discarded. However, since our fitness can only compare individuals among a given generation, its evolution over time does not reflect the global improvement of individuals. This prompted us to perform a post-mortem analysis of our individuals by making the best of each generations of a given run fight each other, highlighting a progressive improvement of our individuals, as shown in Figure 7.4. In particular, the logarithmic shape of the curve goes well with the idea that the efforts required to overcome one's opponents are greater and greater as the simplest strategies get commonly countered.

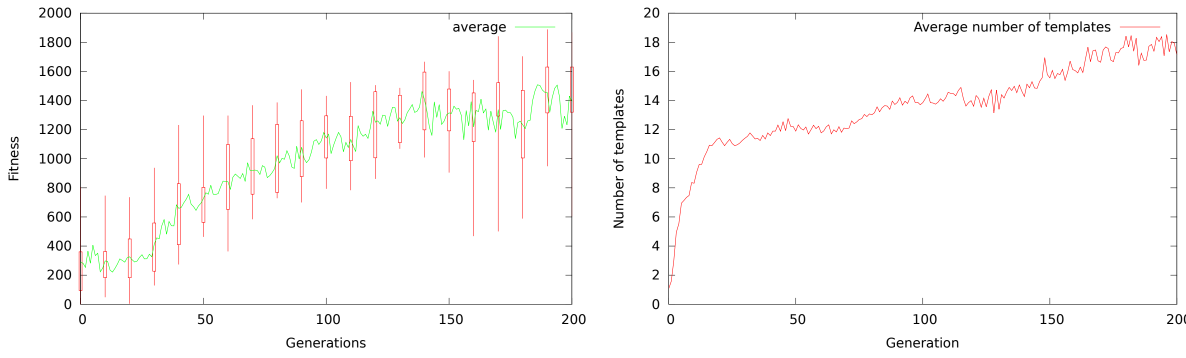


Figure 7.4: Top: average *a posteriori* fitness of the best individuals, as well as minimum, maximum, first and third quartiles. While noisy, the curve still shows an increasing trend similar to that of a logarithm. Bottom: the average number of templates in individuals over generations in a typical run. The trend is similar to that of the fitness, showing that bloating stays within acceptable limits.

7.5.1 Cheating

The easiest, and thus first strategy evolved is actual cheating. Since they have references to what each other will play, and continuous access to current concentrations, the individuals monitor the action of their opponent and try to play accordingly. A minimal example is shown on Figure 7.2. Cheating can be of two kinds: either using a direct connection ("if my opponent plays rock, I will play paper"), or an inhibition ("if my opponent plays rock, I will not play scissors"). Cheating leads in some cases to the apparition of oscillatory behaviors, as both individuals are both trying to play the winning move.

7.5.2 Defense mechanisms

Once cheating appears, it quickly spreads among the whole population, either by cross-over, elimination of individuals which could not adapt, of by parallel discovery of the mechanism. From there on, the only way to improve is to develop mechanisms against the other cheater's spying while at the same time improving the monitoring of its current move. Many defenses where expressed among the evolved individuals, but can mainly be separated into five categories: noise generators, stealth, feint, concentration comparators and fold change detectors. Representatives of all those categories are shown in Figure 7.5.

Noise generators are the easiest form of defense. Since it is fair to assume that the opponent will monitor at least two move sequences to decide its own next move, a simple yet efficient way

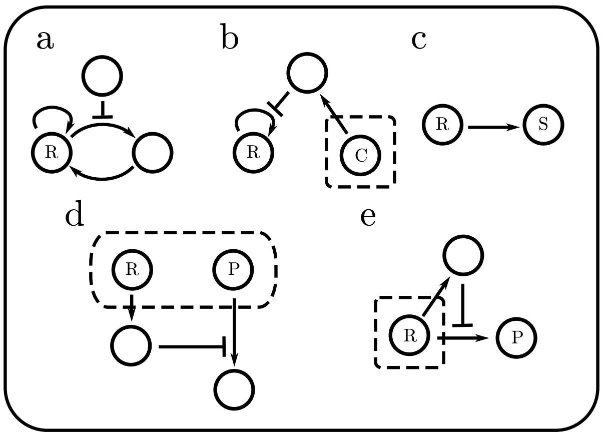


Figure 7.5: Basic mechanisms observed in individuals. (*a.*) Noise generation with two activation level. When the additional path is inhibited, the main sequence will still have a high concentration, but not high enough to be this turn's move. (*b.*) A given move's concentration is kept low for some time by being inhibited by the clock sequence C. (*c.*) A very simple feint: while pretending to play rock (the sequence R) has a non-zero concentration), the individual is actually playing scissors (S), which would win against the expected reaction of the opponent. This mechanism is often decorated with various other systems to balance the concentrations of one sequence relatively to the other. (*d.*) Simple comparison mechanism. The reaction path from the opponent's move will only be activated if the concentration of paper (P) is high enough, compared to the concentration of rock (R). (*e.*) A fold change detector, allowing the monitoring of the increase in the concentration of the rock (R) sequence of the opponent. Often, the detection will happen after a first amplification of the monitored signal.

to keep it off track is to continuously generate all sequences. This is a valid action, since only the highest sequence decides which move is played. Having a weak autocatalytic connection is enough, as long as there is a way for the other sequences to become lower (remember that an individual has to be able to play all moves to have a good fitness). Often, such sequence will have an additional catalytic loop using an additional sequence. This loop is only activated when this sequences is supposed to be played. This simple mechanism allows the individual to have multiple activation levels (by opposition to just "on" and "off"), with a better control on the final concentration of the target sequence rather than using activation mechanisms from different possibly not trustworthy part of the system.

Stealth is the complementary of noise generation. Instead of hiding one's true move among decoys, it is kept at a concentration as near to zero as possible until the last moment. This technique relies on monitoring the clock sequence, since timing is extremely important. The clock sequence is used to generate a large amount of timer, which in turn inhibits a specific move. If the inhibition is stable enough, the target sequence will be kept low until the timer has been degraded. If the delay is not long enough, the opponent will still have time to read and adapt. On the other hand, if the delay is too long, the move will not be valid. Part of the system dedicated to this mechanism seems to be very stable over generations, since it is based on a delicate balancing of parameters where any change can prove deadly.

Feint resembles closely the previous two strategies, but uses a different structure. In this case, the individual spoofs a specific move (say “rock”), but this very move also activates the generation of the real move (for instance “scissors”), often through a long activation path to generate delay. It relies on the fact that the opponent will try to adapt to the perceived move, and won’t be able to react in time to the change. The system may be reset by the clock, or by a change in the opponent’s perceived move.

As the direct monitoring of sequences became less and less reliable, structures to compare absolute concentrations as well as detect sudden modifications became more and more common. Concentration comparison is done through the inhibition of a reaction path if its activation is not strong enough compared to the reference. Since this inhibition originates from the monitoring of another sequence, the first pathway is activated only if the first sequence has a higher concentration. Of course, by tuning the strength of pathways and inhibition, it is possible to have more specific control over the targeted ratio between the two sequences. For instance, it would be possible to slightly modify the system to inhibit the reaction path only if the compared sequence has a concentration multiple times higher than the reference sequence. This defense mechanism is used to counter noise generators and feints.

The last technique commonly spread among individuals is a way to detect concentration increase. While concentration comparison is able to detect that a stealthy move is being played, it is only able to do so once the move became dominant (which, if the other player is timing right, should be too late). However, by using a monitoring coupled with incoherent feedforward, individuals are capable of detecting rapid variations in concentration, which would be a sign that their opponent is about to switch their move. Some individuals also *pretended* to switch their move to throw such defense technique off guard, but this was quickly countered by a mix of both direct comparison and incoherent feedforward.

7.5.3 Memory vs cheating

Quite early on, individuals with a basic memory, such as the bistable from Figure 7.1, appear in the population. However, those individuals were too “naive” in the sense that they had no defense against cheaters. Moreover, cheating requires about the same amount of mutations to appear, or even less if partial (that is, the individual can read *some* moves, but not all). For this reason, it seems that it is much more advantageous for individuals to focus only on attack and defense. This prevented the reapparition of memory in later generation, leading to purely reactive individuals.

7.5.4 The arms race

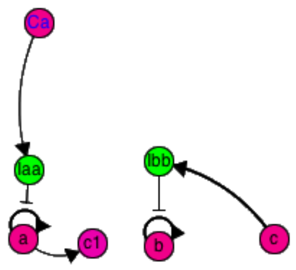
Looking at individuals over time shows the apparitions of the different cheating and defense mechanisms over time, with a noticeable complexification of the best individuals. Figure 7.6 shows such individuals at different times of a specific run, highlighting the apparition of various mechanisms.

The logical conclusion of this evolution strategy is that individuals with high fitness in a given generation have very little, or even no structures that are not related to cheating and defeating. Even when they exist, such structures are mutated during the next few generations to serve some attack or defense purpose. We performed an *a posteriori* evaluation of the fitness to check whether this increase in individuals size was indeed justified or only bloating. By performing this evaluation, we get a sense of the improvement of individuals over time that cannot be deduced from the lexicographic fitness used for evolution, since the later one only compares individuals from a given generation. The fitness itself is computed by making the best individual of all generations fight each other and score points in the same fashion than in the second part of the lexicographic fitness.

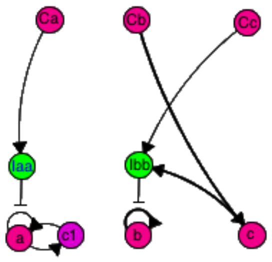
The trend of the *a posteriori* fitness also implies that there is no cyclic effect. While the lexicographic fitness guarantees that all individuals have the capacity of playing any move given the right conditions, there could be more advanced strategy displaying such cyclic dynamics. For instance, individuals using stealth are beaten by individuals using incoherent feedforwards, which could have been, in turn, beaten by another strategy that is weak against stealth. Since the fitness increase is monotonic (if we ignore the noise), we can conclude that the arms race is open-ended, with complexification of individuals the only possible way to improve.

We could also note that the arms race pushes individuals to perform well within their own ecosystem, but not always optimally. For instance, the individual from generation 122 in Figure 7.6 only defends against stealthy changes in the concentration of "paper", leaving it open to the exact same strategy, if performed on another move. However, it is easy for a human designer to take inspiration from those modules to create an "optimal" player.

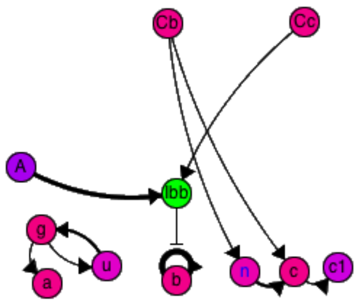
We also performed a negative control where cheating was explicitly forbidden. Instead, agents had two input signals: one for "victory" and the other for "defeat". Obviously, no specific structure was encoded in the initial individuals, so they had to learn through multiple generation



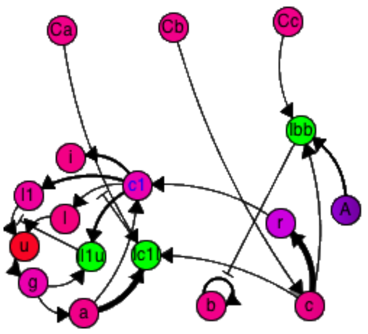
Generation 10: partial cheating.



Generation 14: complete cheater.



Generation 109: stealth. The clock sequence (here designated A) hides a move (b).



Generation 122: fold change detector. The sequence c both activates and inhibits the creation of a. However, the activation path is longer than the inhibition path, meaning that a (rock) is only activated by this module if the concentration of c (scissors) is decreasing. Since c is directly linked to the opponent's b (paper), this individual is protected against stealthy play of b.

Figure 7.6: Individuals generated during a run. The color of activation nodes indicates their stability, going from red (very unstable) to blue (very stable). Green nodes are inhibitors. The notation for the moves rock, paper and scissors is respectively a, b and c. References to the opponent's sequences are designated by a leading C. A represents the clock.

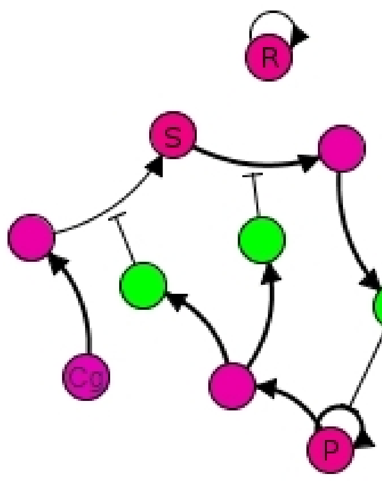


Figure 7.7: A non-cheating individual. R, P and S stands respectively for Rock, Paper and Scissors. Cg correspond to loosing a hand. The strategy of this individual is as follow: play Paper, if the hand is lost or there is a draw, play scissors for a while (note the almost bistable structure). When the priority of both Scissors and Paper are identical, the default is Rock.

the meaning of those signals. As was originally expected, we can observe the emergence of storing structures. Such structures allow the individuals to remember what were the previous moves and decide what to do next. However, as was noted before, such structures are also harder to evolve, making the progression much slower. Figure 7.7 shows a typical well-performing individual evolved this way.

7.6 Conclusion

In this work, we demonstrated how an evolutionary strategy could be used to create complex systems. For this, we introduced bioNEAT, a modified version of NEAT designed to evolve chemical reaction networks from the DNA toolbox. We chose to focus on finding a good strategy to play the well-studied game of Rock–Paper–Scissors. Our first hope was to observe the emergence of memory to allow non-trivial strategies However, the very rules, derived from experimental settings, we set for the game prevented this memory mechanism from being efficient. Instead, increasingly complex cheating seemed to be the best answer. Nonetheless, this is not the only thing we learned from this exercise. While having DNA systems compete against each other and evolve new (cheating) strategies can be a goal in itself, the systems evolved along the way gave us also more insight about DNA computing systems. In particular, it was possible to observe the emergence of particular structures with interesting dynamics, which may prove useful to a human trying to develop DNA systems, like with the libraries of [177]. It could be also interesting to make individuals compete against a human designed “optimal” cheater and see if they can evolve even more advanced strategies to counter it. Furthermore, since the DNA toolbox mimic the behavior of gene regulatory circuits [67], an open question would be whether those mechanisms appear in real life or if they are only valid in the toolbox. Also, it would be interesting to extend the current systems to take into account reaction-diffusion and be able to play more complex games. There is little doubt that such systems will have their own share of remarkable mechanisms.

Chapter 8

Conclusion

DNA computing has been a very active and exciting field for the past few years. The size of DNA computing systems is increasing at an exponential pace similar to the Moore law [50]. Now that impressive achievements have been obtained with various paradigms [10, 41, 67, 11], we have to push the boundaries of what is possible to realize *in vitro*. Automation of experiments, well understood building blocks and fast design capabilities are all parts of what can bring DNA computing to the next stage.

We focused on the design and prototyping aspect, providing both new building blocks and computer assistance. The DNA toolbox proved to be the perfect base framework: it is capable to display many different dynamical behaviors while having easy to model and combine modules. Additionally, the DNA toolbox represents a way to simulate biochemically gene regulatory networks. While those processes are extremely complex and hard to manipulate, the DNA toolbox offers a sandbox to test biological hypothesis.

Our model of the DNA toolbox, which is a formalization and extension of the one appearing in the work of Padirac [43], was used as a base for design automation of DNA computing systems. We presented two different design approaches. DACCAD allows user to create the systems quickly, either through its graphical interface or by compiling into a graph representation. On the other hand, we used an evolutionary approach, called BioNEAT, where the user only defines a specific objective and the algorithm then optimize both the structure (templates used in the system) and the chemical and biological parameters to approach this goal as close as possible. We showed, by implementing the rules of the Rock-Paper-Scissor game, that this approach was valid even with abstract objectives. Moreover, thanks to their common representation of the DNA toolbox, it is possible (and even recommended, for a real-life application) to go back and forth between the user-guided approach and the evolutionary approach. The user can give a little push in the right direction, when necessary, while the automated optimization can be used to improve specific subsystems.

We also added new building blocks to the DNA toolbox, with the objective to increase the ease to create new systems. This was done both by creating a new low level module, the delay gate, that can integrate seamlessly with the DNA toolbox and by exhibiting building patterns with various properties. Those last patterns were found through constant use of the paradigm, mainly relying on DACCAD, and by automatic discovery from BioNEAT. The delay gate has still some experimental hiccups, and additional efforts are needed before it can be formally integrated with the DNA toolbox.

The next step is now to integrate many different paradigms together. The possibility to use the SBML file format gives us a nice platform for combining systems and create complex applications. In particular, there are many model of the cell available, and it would be interesting to simulate how the DNA toolbox is behaving in those environments. Additionally, with the recent push toward two dimensional systems, the appearance of smart materials or DNA-based distributed computing is closer than ever.

Yet, additional tools are required of those applications. The predator-prey system [119] shows great potential and stability, but cannot be manipulated by the current model. In particular, this system requires "signal" species to modify other species, which breaks the rules we assumed to create the graph representation in Chapter 2. One possibility could be to relax the "graph" conditions, and instead use more advanced structures, such as 2-categories [178]. Instead, the best approach might be to have an adaptive model coupled with an automated experimental setting, allowing us to free ourselves from *ab initio* limitations.

Nonetheless, taking a step back from the technology, it is amazing to see all the possibilities that exist in DNA computing. As soon as we could explore with ease the space of possible behaviors, it was fairly easy to find impressive systems. It makes one wonder what will be possible to achieve when moving away from well-mixed 0-dimensional systems to more complex structures.

At the 18th conference on DNA computing and molecular programming (DNA 18), a panel was dedicated to this question: what will be the "killer app" of DNA computing. In other words, what is the application only DNA computing can do, or at least the application DNA computing can do overwhelmingly better than any other approach. Hopefully, the availability of better and more diverse designing possibilities will enable the emergence of such application. To finish, I want to mention that playing is as good a way to drive people forward as another, and "serious gaming" [179] might be also a valid venue of investigation. Hopefully, having demonstrated the possibility to play various games with the DNA toolbox (Mastermind, Rock–Paper–Scissors and DNA wars[1]), along with already existing games [174] will attract an even more diverse crowd to DNA computing.

[1]Presented at an impromptu session of the DNA 18 conference

Bibliography

[1] Douglas Adams. *The Hitchhiker's Guide to the Galaxy*. 1979.

[2] Alan M Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 42(2):230–265, 1936.

[3] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, 1996.

[4] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*, volume 4. Oxford university press New York, 1999.

[5] Alan Mathison Turing. The chemical basis of morphogenesis. *Bulletin of mathematical biology*, 52(1):153–197, 1990.

[6] David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, 2010.

[7] Anne Condon, Alan J Hu, Ján Maňuch, and Chris Thachuk. Less haste, less waste: on recycling and its limits in strand displacement systems. *Interface focus*, 2(4):512–521, 2012.

[8] Georg Seelig, David Soloveichik, David Yu Zhang, and Erik Winfree. Enzyme-free nucleic acid logic circuits. *science*, 314(5805):1585–1588, 2006.

[9] Erik Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, 1998.

[10] Lulu Qian and Erik Winfree. Scaling up digital circuit computation with dna strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.

[11] A. Padirac, T. Fujii, and Rondelez Y. Bottom-up construction of in vitro switchable memories. *Proceedings of the National Academy of Sciences*, 109(47):E3212–E3220, 2012.

[12] Jongmin Kim, John Hopfield, and Erik Winfree. Neural network computation by in vitro transcriptional circuits. In *Advances in neural information processing systems*, pages 681–688, 2004.

[13] Leonard M Adleman. Molecular computation of solutions to combinatorial problems. *SCIENCE-NEW YORK THEN WASHINGTON-*, pages 1021–1021, 1994.

[14] Ravinderjit S Braich, Nickolas Chelyapov, Cliff Johnson, Paul WK Rothemund, and Leonard Adleman. Solution of a 20-variable 3-sat problem on a dna computer. *Science*, 296(5567):499–502, 2002.

[15] Luca Cardelli. From processes to odes by chemistry. In *Fifth Ifip International Conference On Theoretical Computer Science–Tcs 2008*, pages 261–281. Springer, 2008.

[16] Marcelo O Magnasco. Chemical kinetics is turing universal. *Physical Review Letters*, 78(6):1190–1193, 1997.

[17] Anastasia Deckard and Herbert M Sauro. Preliminary studies on the in silico evolution of biochemical networks. *ChemBioChem*, 5(10):1423–1431, 2004.

[18] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994.

[19] Peter Dittrich, Jens Ziegler, and Wolfgang Banzhaf. Artificial chemistriesa review. *Artificial life*, 7(3):225–275, 2001.

[20] AN Zaikin and AM Zhabotinsky. Concentration wave propagation in two-dimensional liquid-phase self-oscillating system. *Nature*, 225(5232):535–537, 1970.

[21] Nathaniel Virgo and Takashi Ikegami. Autocatalysis before enzymes: The emergence of prebiotic chain reactions. In *Advances in Artificial Life, ECAL*, volume 12, pages 240–247, 2013.

[22] Andreas Gustafsson. *Egypt.* Gson, version 1.10 edition, 2011.

[23] Qing Zhou, Hiram Chipperfield, Douglas A Melton, and Wing Hung Wong. A gene regulatory network in mouse embryonic stem cells. *Proceedings of the National Academy of Sciences*, 104(42):16438–16443, 2007.

[24] James D Watson, Francis HC Crick, et al. Molecular structure of nucleic acids. *Nature*, 171(4356):737–738, 1953.

[25] Paul WK Rothemund. Folding dna to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.

[26] Shelley FJ Wickham, Masayuki Endo, Yousuke Katsuda, Kumi Hidaka, Jonathan Bath, Hiroshi Sugiyama, and Andrew J Turberfield. Direct observation of stepwise movement of a synthetic molecular transporter. *Nature nanotechnology*, 6(3):166–169, 2011.

[27] Dage Liu, Sung Ha Park, John H Reif, and Thomas H LaBean. Dna nanotubes self-assembled from triple-crossover tiles as templates for conductive nanowires. *Proceedings of the National Academy of Sciences of the United States of America*, 101(3):717–722, 2004.

[28] Sung Ha Park, Constantin Pistol, Sang Jung Ahn, John H Reif, Alvin R Lebeck, Chris Dwyer, and Thomas H LaBean. Finite-size, fully addressable dna tile lattices formed by hierarchical assembly procedures. *Angewandte Chemie*, 118(5):749–753, 2006.

[29] Bryan Wei, Mingjie Dai, and Peng Yin. Complex shapes self-assembled from single-stranded dna tiles. *Nature*, 485(7400):623–626, 2012.

[30] Chengde Mao, Weiqiong Sun, Zhiyong Shen, and Nadrian C Seeman. A nanomechanical device based on the b–z transition of dna. *Nature*, 397(6715):144–146, 1999.

[31] Bernard Yurke, Andrew J Turberfield, Allen P Mills, Friedrich C Simmel, and Jennifer L Neumann. A dna-fuelled molecular machine made of dna. *Nature*, 406(6796):605–608, 2000.

[32] Hao Yan, Xiaoping Zhang, Zhiyong Shen, and Nadrian C Seeman. A robust dna mechanical device controlled by hybridization topology. *Nature*, 415(6867):62–65, 2002.

[33] Banani Chakraborty, Ruojie Sha, and Nadrian C Seeman. A dna-based nanomechanical device with three robust states. *Proceedings of the National Academy of Sciences*, 105(45):17245–17249, 2008.

[34] Jong-Shik Shin and Niles A Pierce. A synthetic dna walker for molecular transport. *Journal of the American Chemical Society*, 126(35):10834–10835, 2004.

[35] Ye Tian, Yu He, Yi Chen, Peng Yin, and Chengde Mao. A dnazyme that walks processively and autonomously along a one-dimensional track. *Angewandte Chemie International Edition*, 44(28):4355–4358, 2005.

[36] Renjun Pei, Steven K Taylor, Darko Stefanovic, Sergei Rudchenko, Tiffany E Mitchell, and Milan N Stojanovic. Behavior of polycatalytic assemblies in a substrate-displaying matrix. *Journal of the American Chemical Society*, 128(39):12693–12699, 2006.

[37] Ronald R Breaker. Dna aptamers and dna enzymes. *Current opinion in chemical biology*, 1(1):26–31, 1997.

[38] Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M LeProust, Botond Sipos, and Ewan Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized dna. *Nature*, 2013.

[39] Joris JM Gillis and Jan Van den Bussche. A formal model for databases in dna. In *Algebraic and numeric biology*, pages 18–37. Springer, 2012.

[40] Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from dna. *Nature nanotechnology*, 8(10):755–762, 2013.

[41] Lulu Qian, Erik Winfree, and Jehoshua Bruck. Neural network computation with dna strand displacement cascades. *Nature*, 475(7356):368–372, 2011.

[42] Anthony J Genot, Teruo Fujii, and Yannick Rondelez. Scaling down dna circuits with competitive neural networks. *Journal of The Royal Society Interface*, 10(85), 2013.

[43] Adrien Padirac. *Tailoring spatiotemporal dynamics with DNA circuits*. PhD thesis, Université Claude Bernard Lyon I, 2012.

[44] Ebbe S Andersen, Mingdong Dong, Morten M Nielsen, Kasper Jahn, Ramesh Subramani, Wael Mamdouh, Monika M Golas, Bjoern Sander, Holger Stark, Cristiano LP Oliveira, et al. Self-assembly of a nanoscale dna box with a controllable lid. *Nature*, 459(7243):73–76, 2009.

[45] Karen Scida, Bingling Li, Andrew D Ellington, and Richard M Crooks. Dna detection using origami paper analytical devices. *Analytical Chemistry*, 2013.

[46] Mark Schena, Dari Shalon, Ronald W Davis, and Patrick O Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270(5235):467–470, 1995.

[47] Xi Chen and Andrew D Ellington. Shaping up nucleic acid computation. *Current opinion in biotechnology*, 21(4):392–400, 2010.

[48] Adrien Padirac, Teruo Fujii, and Yannick Rondelez. Nucleic acids for the rational design of reaction circuits. *Current opinion in biotechnology*, 2012.

[49] Yangyang Yang, Masayuki Endo, Kumi Hidaka, and Hiroshi Sugiyama. Photo-controllable dna origami nanostructures assembling into predesigned multiorientational patterns. *Journal of the American Chemical Society*, 134(51):20645–20653, 2012.

[50] Robert Carlson. The changing economics of dna synthesis. *Nature biotechnology*, 27(12):1091, 2009.

[51] Wilma K Olson and Victor B Zhurkin. Modeling dna deformations. *Current opinion in structural biology*, 10(3):286–297, 2000.

[52] Pablo D Dans, Ari Zeida, Matias R Machado, and Sergio Pantano. A coarse grained model for atomic-detailed dna simulations with explicit electrostatics. *Journal of Chemical Theory and Computation*, 6(5):1711–1725, 2010.

[53] Thomas E Ouldridge. *Springer Theses: Coarse-Grained Modelling of Dna and Dna Self-Assembly.* Springer, 2012.

[54] Jonathan PK Doye, Thomas E Ouldridge, Ard A Louis, Flavio Romano, Petr Šulc, Christian Matek, Benedict EK Snodin, Lorenzo Rovigatti, John S Schreck, Ryan M Harrison, et al. Coarse-graining dna for simulations of dna nanotechnology. *Physical Chemistry Chemical Physics*, 15(47):20395–20414, 2013.

[55] Yonggang Ke, Luvena L Ong, William M Shih, and Peng Yin. Three-dimensional structures self-assembled from dna bricks. *Science*, 338(6111):1177–1183, 2012.

[56] Carlos Ernesto Castro, Fabian Kilchherr, Do-Nyun Kim, Enrique Lin Shiao, Tobias Wauer, Philipp Wortmann, Mark Bathe, and Hendrik Dietz. A primer to scaffolded dna origami. *Nature methods*, 8(3):221–229, 2011.

[57] Nicholas R Markham and Michael Zuker. Dinamelt web server for nucleic acid melting prediction. *Nucleic acids research*, 33(suppl 2):W577–W581, 2005.

[58] Jeffrey W Nelson and Ignacio Tinoco Jr. Comparison of the kinetics of ribo-, deoxyribo- and hybrid oligonucleotide double-strand formation by temperature-jump kinetics. *Biochemistry*, 21(21):5289–5295, 1982.

[59] David Yu Zhang, Andrew J Turberfield, Bernard Yurke, and Erik Winfree. Engineering entropy-driven reactions and networks catalyzed by dna. *Science*, 318(5853):1121–1125, 2007.

[60] Anthony J Genot, David Yu Zhang, Jonathan Bath, and Andrew J Turberfield. Remote toehold: a mechanism for flexible control of dna hybridization kinetics. *Journal of the American Chemical Society*, 133(7):2177–2182, 2011.

[61] David Yu Zhang and Erik Winfree. Control of dna strand displacement kinetics using toehold exchange. *Journal of the American Chemical Society*, 131(47):17303–17314, 2009.

[62] Yingfu Li, Ronald Geyer, and Dipankar Sen. Recognition of anionic porphyrins by dna aptamers. *Biochemistry*, 35(21):6911–6922, 1996.

[63] Louis C Bock, Linda C Griffin, John A Latham, Eric H Vermaas, and John J Toole. Selection of single-stranded dna molecules that bind and inhibit human thrombin. 1992.

[64] Sam Rowels Erik Winfree Richard Burgoyne, Nickolas V Chelyapov, Myron F Goodman, Paul WK Rothemund, and Leonard M Adleman. A sticker based model for dna computation. *DNA Based Computers Two*, 44:1, 1999.

[65] Lulu Qian, David Soloveichik, and Erik Winfree. Efficient turing-universal computation with dna polymers. In *DNA computing and molecular programming*, pages 123–140. Springer, 2011.

[66] Lulu Qian and Erik Winfree. A simple dna gate motif for synthesizing large-scale circuits. *Journal of the Royal Society Interface*, 8(62):1281–1297, 2011.

[67] K. Montagne, R. Plasson, Y. Sakai, T. Fujii, and Y. Rondelez. Programming an in vitro dna oscillator using a molecular networking strategy. *Molecular systems biology*, 7(1), 2011.

[68] Yaakov Benenson, Tamar Paz-Elizur, Rivka Adar, Ehud Keinan, Zvi Livneh, and Ehud Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414(6862):430–434, 2001.

[69] John A Rose, Russell J Deaton, Masami Hagiya, and Akira Suyama. Pna-mediated whiplash pcr. In *DNA Computing*, pages 104–116. Springer, 2002.

[70] Gijs JL Wuite, Steven B Smith, Mark Young, David Keller, and Carlos Bustamante. Single-molecule studies of the effect of template tension on t7 dna polymerase activity. *Nature*, 404(6773):103–106, 2000.

[71] Piet Herdewijn and Philippe Marliere. Toward safe genetically modified organisms through the chemical diversification of nucleic acids. *Chemistry & biodiversity*, 6(6):791–808, 2009.

[72] Vitor B Pinheiro and Philipp Holliger. The xna world: progress towards replication and evolution of synthetic genetic polymers. *Current Opinion in Chemical Biology*, 16(3):245–252, 2012.

[73] Justin K Ichida, Allen Horhota, Keyong Zou, Larry W McLaughlin, and Jack W Szostak. High fidelity tna synthesis by therminator polymerase. *Nucleic acids research*, 33(16):5219–5225, 2005.

[74] Sanjay K Singh, Alexei A Koshkin, Jesper Wengel, and Poul Nielsen. Lna (locked nucleic acids): synthesis and high-affinity nucleic acid recognition. *Chemical Communications*, (4):455–456, 1998.

[75] Hidehito Urata, Emiko Ogura, Keiko Shinohara, Yoshiaki Ueda, and Masao Akagi. Synthesis and properties of mirror-image dna. *Nucleic acids research*, 20(13):3325–3332, 1992.

[76] Nicole C Hauser, Rafael Martinez, Anette Jacob, Steffen Rupp, Jörg D Hoheisel, and Stefan Matysiak. Utilising the left-helical conformation of l-dna for analysing different marker types on a single universal microarray platform. *Nucleic acids research*, 34(18):5101–5111, 2006.

[77] Alexei A Koshkin, Poul Nielsen, Michael Meldgaard, Vivek K Rajwanshi, Sanjay K Singh, and Jesper Wengel. Lna (locked nucleic acid): an rna mimic forming exceedingly stable lna: Lna duplexes. *Journal of the American Chemical Society*, 120(50):13252–13253, 1998.

[78] Ramon Kranaster and Andreas Marx. Engineered dna polymerases in biotechnology. *ChemBioChem*, 11(15):2077–2084, 2010.

[79] Ichiro Hirao, Yoko Harada, Michiko Kimoto, Tsuneo Mitsui, Tsuyoshi Fujiwara, and Shigeyuki Yokoyama. A two-unnatural-base-pair system toward the expansion of the genetic code. *Journal of the American Chemical Society*, 126(41):13298–13305, 2004.

[80] Ichiro Hirao, Michiko Kimoto, and Rie Yamashige. Natural versus artificial creation of base pairs in dna: origin of nucleobases from the perspectives of unnatural base pair studies. *Accounts of Chemical Research*, 45(12):2055–2065, 2012.

[81] Michael W Davidson, Teresa E Strzelecka, and Randolph L Rill. Multiple liquid crystal phases of dna at high concentrations. *Nature*, 331:457–460, 1988.

[82] Richard J Lipton et al. Dna solution of hard computational problems. *Science*, 268(5210):542–545, 1995.

[83] Kensaku Sakamoto, Hidetaka Gouzu, Ken Komiya, Daisuke Kiga, Shigeyuki Yokoyama, Takashi Yokomori, and Masami Hagiya. Molecular computation by dna hairpin formation. *Science*, 288(5469):1223–1226, 2000.

[84] Justin Werfel and Radhika Nagpal. Extended stigmergy in collective construction. *Intelligent Systems, IEEE*, 21(2):20–28, 2006.

[85] Harish Chandran, Nikhil Gopalkrishnan, Bernard Yurke, and John Reif. Meta-dna: synthetic biology via dna nanostructures and hybridization reactions. *Journal of The Royal Society Interface*, 9(72):1637–1653, 2012.

[86] Jean-Christophe Galas, Anne-Marie Haghiri-Gosnet, and André Estévez-Torres. A nanoliter-scale open chemical reactor. *Lab on a Chip*, 13(3):415–423, 2013.

[87] Leroy Cronin. Synthetic biology manipulations in 3d printed wet-ware. In *Advances in Artificial Life, ECAL*, volume 12, pages 1142–1142, 2013.

[88] A. Vlandas, A. Padirac, A. Estevez-Torres, and Y. Rondelez. Switching a bistable dna circuit electrically. In *Proceedings of the International Conference on DNA Computing and Molecular Programming*, volume 19, pages 76–76, 2013.

[89] Yannick Rondelez. Competition for catalytic resources alters biological network dynamics. *Physical review letters*, 108(1):018102, 2012.

[90] Michael Hucka, Andrew Finney, Herbert M Sauro, Hamid Bolouri, John C Doyle, Hiroaki Kitano, Adam P Arkin, Benjamin J Bornstein, Dennis Bray, Athel Cornish-Bowden, et al. The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.

[91] Andrew Finney and Michael Hucka. Systems biology markup language: Level 2 and beyond. *Biochemical Society Transactions*, 31(6):1472–1473, 2003.

[92] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 312–317. IEEE, 1996.

[93] Joseph N Zadeh, Conrad D Steenberg, Justin S Bois, Brian R Wolfe, Marshall B Pierce, Asif R Khan, Robert M Dirks, and Niles A Pierce. Nupack: analysis and design of nucleic acid systems. *Journal of computational chemistry*, 32(1):170–173, 2011.

[94] Nicholas R Markham and Michael Zuker. Unafold. In *Bioinformatics*, pages 3–31. Springer, 2008.

[95] Elisa Franco, Eike Friedrichs, Jongmin Kim, Ralf Jungmann, Richard Murray, Erik Winfree, and Friedrich C Simmel. Timing molecular motion and production with a synthetic transcriptional clock. *Proceedings of the National Academy of Sciences*, 108(40):E784–E793, 2011.

[96] Matthew R Lakin, Simon Youssef, Filippo Polo, Stephen Emmott, and Andrew Phillips. Visual dsd: a design and analysis tool for dna strand displacement systems. *Bioinformatics*, 27(22):3211–3213, 2011.

[97] Nathanael Aubert, Clement Moscat, Teruo Fujii, Masami Hagiya, and Yannick Rondelez. Computer assisted design for scaling up systems based on dna reaction networks. *Natural Computing*, Accepted.

[98] Vadim A Vasiliskov, Dmitry V Prokopenko, and Andrei D Mirzabekov. Parallel multiplex thermodynamic analysis of coaxial base stacking in dna duplexes by oligodeoxyribonucleotide microchips. *Nucleic acids research*, 29(11):2303–2313, 2001.

[99] James Dickson Murray. *Mathematical biology*, volume 1. springer, 2002.

[100] Raphaël Plasson and Yannick Rondelez. Synthetic biochemical dynamic circuits. *Multiscale Analysis and Nonlinear Dynamics: From Genes to the Brain*, pages 113–145, 2013.

[101] A. Baccouche, K. Montagne, A. Padirac, and Y. Rondelez. Dynamic dna-toolbox reaction circuits: a walkthrough. *Methods, submitted*, 2013.

[102] Matthew R Lakin, David Parker, Luca Cardelli, Marta Kwiatkowska, and Andrew Phillips. Design and analysis of dna strand displacement devices using probabilistic model checking. *Journal of the Royal Society Interface*, 9(72):1470–1485, 2012.

[103] Wilson W Wong, Tony Y Tsai, and James C Liao. Single-cell zeroth-order protein degradation enhances the robustness of synthetic oscillator. *Molecular systems biology*, 3(1), 2007.

[104] Dmitrii V Pyshnyi and Eugenia M Ivanova. The influence of nearest neighbours on the efficiency of coaxial stacking at contiguous stacking hybridization of oligodeoxyribonucleotides. *Nucleosides, Nucleotides and Nucleic Acids*, 23(6-7):1057–1064, 2004.

[105] Peter Yakovchuk, Ekaterina Protozanova, and Maxim D Frank-Kamenetskii. Base-stacking and base-pairing contributions into thermal stability of the dna double helix. *Nucleic acids research*, 34(2):564–574, 2006.

[106] Ekaterina Protozanova, Peter Yakovchuk, and Maxim D Frank-Kamenetskii. Stacked–unstacked equilibrium at the nick site of dna. *Journal of molecular biology*, 342(3):775–785, 2004.

[107] Lubert Stryer. Fluorescence energy transfer as a spectroscopic ruler. *Annual review of biochemistry*, 47(1):819–846, 1978.

[108] Adrien Padirac, Teruo Fujii, and Yannick Rondelez. Quencher-free multiplexed monitoring of dna reaction circuits. *Nucleic acids research*, 40(15):e118–e118, 2012.

[109] Jifeng Qian, Tanya M Ferguson, Deepali N Shinde, Alissa J Ramírez-Borrero, Arend Hintze, Christoph Adami, and Angelika Niemz. Sequence dependence of isothermal dna amplification via expar. *Nucleic acids research*, 40(11):e87–e87, 2012.

[110] Inc. Wolfram Research. *Mathematica*. Wolfram Research, Inc., version 8.0 edition, 2010.

[111] John SantaLucia. A unified view of polymer, dumbbell, and oligonucleotide dna nearest-neighbor thermodynamics. *Proceedings of the National Academy of Sciences*, 95(4):1460–1465, 1998.

[112] Nathanael Aubert, Yannick Rondelez, Teruo Fujii, and Masami Hagiya. Enforcing logical delays in dna computing systems. *Natural Computing*, Accepted.

[113] Yaakov Benenson, Binyamin Gil, Uri Ben-Dor, Rivka Adar, and Ehud Shapiro. An autonomous molecular computer for logical control of gene expression. *Nature*, 429(6990):423–429, 2004.

[114] Leslie Lamport. Concurrent reading and writing. *Communications of the ACM*, 20(11):806–811, 1977.

[115] Yohsuke Bansho, Norikazu Ichihashi, Yasuaki Kazuta, Tomoaki Matsuura, Hiroaki Suzuki, and Tetsuya Yomo. Importance of parasite rna species repression for prolonged translation-coupled rna self-replication. *Chemistry & Biology*, 19(4):478–487, 2012.

[116] Chris Thachuk and Anne Condon. Space and energy efficient computation with dna strand displacement systems. In *DNA Computing and Molecular Programming*, pages 135–149. Springer, 2012.

[117] Kevin Montagne, Raphael Plasson, Adrien Padirac, Teruo Fujii, and Yannick Rondelez. A toolbox to build time-responsive in vitro dna networks. In *Oral presentation, 17th International Conference of DNA Computing and Molecular Programming*, 2011.

[118] Anthony J Genot, Teruo Fujii, and Yannick Rondelez. Computing with competition in biochemical networks. *Physical review letters*, 109(20):208102, 2012.

[119] Teruo Fujii and Yannick Rondelez. Predator–prey molecular ecosystems. *ACS nano*, 7(1):27–34, 2012.

[120] David Whitcombe, Jane Theaker, Simon P Guy, Tom Brown, and Steve Little. Detection of pcr products using self-probing amplicons and fluorescence. *Nature biotechnology*, 17(8):804–807, 1999.

[121] Leslie Lamport. Proving the correctness of multiprocess programs. *Software Engineering, IEEE Transactions on*, (2):125–143, 1977.

[122] Renjun Pei, Elizabeth Matamoros, Manhong Liu, Darko Stefanovic, and Milan N Stojanovic. Training a molecular automaton to play a game. *Nature Nanotechnology*, 5(11):773–777, 2010.

[123] Andreas Wagner and David A Fell. The small world inside large metabolic networks. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 268(1478):1803–1810, 2001.

[124] Michael S Branicky. Multiple lyapunov functions and other analysis tools for switched and hybrid systems. *Automatic Control, IEEE Transactions on*, 43(4):475–482, 1998.

[125] John J Tyson, Katherine C Chen, and Bela Novak. Sniffers, buzzers, toggles and blinkers: dynamics of regulatory and signaling pathways in the cell. *Current opinion in cell biology*, 15(2):221–231, 2003.

[126] Ehsan Chiniforooshan, David Doty, Lila Kari, and Shinnosuke Seki. Scalable, time-responsive, digital, energy-efficient molecular circuits using dna strand displacement. In *DNA Computing and Molecular Programming*, pages 25–36. Springer, 2011.

[127] Anthony J Genot, Jonathan Bath, and Andrew J Turberfield. Reversible logic circuits made of dna. *Journal of the American Chemical Society*, 133(50):20080–20083, 2011.

[128] Adrien Padirac, Teruo Fujii, André Estévez-Torres, and Yannick Rondelez. Spatial waves in synthetic biochemical networks. *Journal of the American Chemical Society*, 2013.

[129] Koshi Hasatani, Mathieu Leocmach, Anthony J Genot, André Estévez-Torres, Teruo Fujii, and Yannick Rondelez. High-throughput and long-term observation of compartmentalized biochemical oscillators. *Chemical Communications*, 2013.

[130] Jongmin Kim and Erik Winfree. Synthetic in vitro transcriptional oscillators. *Molecular systems biology*, 7(1), 2011.

[131] Brian C Goodwin. Oscillatory behavior in enzymatic control processes. *Advances in enzyme regulation*, 3:425–437, 1965.

[132] Michael B Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.

[133] Albert-László Barabási and Zoltan N Oltvai. Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–113, 2004.

[134] Stefan Hoops, Sven Sahle, Ralph Gauges, Christine Lee, Jürgen Pahle, Natalia Simus, Mudita Singhal, Liang Xu, Pedro Mendes, and Ursula Kummer. Copasia complex pathway simulator. *Bioinformatics*, 22(24):3067–3074, 2006.

[135] Paul WK Rothemund. Folding dna to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.

[136] Shawn M Douglas, Adam H Marblestone, Surat Teerapittayanon, Alejandro Vazquez, George M Church, and William M Shih. Rapid prototyping of 3d dna-origami shapes with cadnano. *Nucleic acids research*, 37(15):5001–5006, 2009.

[137] Joshua OMadadhain, Danyel Fisher, Padhraic Smyth, Scott White, and Yan-Biao Boey. Analysis and visualization of network data using jung. *Journal of Statistical Software*, 10(2):1–35, 2005.

[138] David Gilbert. The jfreechart class library. *Developer Guide. Object Refinery*, 2002.

[139] E Hairer, SP Nørsett, and G Wanner. Solving ordinary differential equations i. nonstiff problems, 1987.

[140] Q. H. Dinh, N. Aubert, N. Noman, T. Fujii, Y. Rondelez, and H. Iba. An effective method for evolving reaction network in synthetic biochemical systems. *IEEE Transactions on Evolutionary Computation, submitted*, 2013.

[141] Nathanaël Aubert, Quang Huy Dinh, Masami Hagiya, Hitoshi Iba, Teruo Fujii, Nicolas Bredeche, and Yannick Rondelez. Evolution of cheating dna-based agents playing the game of rock-paper-scissors. In *Advances in Artificial Life, ECAL*, volume 12, pages 1143–1150, 2013.

[142] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[143] M. Hagiya, I. Kawamata, and N. Aubert. Towards persistent molecular computers for molecular robots. In *DNA Computing and Molecular Programming 19th International Conference, DNA19, extended abstract*, 2013.

[144] Aryeh Warmflash, Paul Francois, and Eric D Siggia. Pareto evolution of gene networks: an algorithm to optimize multiple fitness objectives. *Physical Biology*, 9(5):056001, 2012.

[145] Steven Muchnick. *Advanced compiler design and implementation*. Morgan Kaufmann, 1997.

[146] Charles E Leiserson, Ronald L Rivest, Clifford Stein, and Thomas H Cormen. *Introduction to algorithms*. The MIT press, 2001.

[147] Brian W Kernighan, Dennis M Ritchie, and Per Ejeklint. *The C programming language*, volume 2. prentice-Hall Englewood Cliffs, 1988.

[148] Luca Cardelli. Strand algebras for dna computing. *Natural Computing*, 10(1):407–428, 2011.

[149] Andrew Phillips and Luca Cardelli. A programming language for composable dna circuits. *Journal of the Royal Society Interface*, 6(Suppl 4):S419–S436, 2009.

[150] Matthew R Lakin, Simon Youssef, Luca Cardelli, and Andrew Phillips. Abstractions for dna circuit design. *Journal of The Royal Society Interface*, 9(68):470–486, 2012.

[151] Peter Cheeseman, Bob Kanefsky, and William M Taylor. Where the really hard problems are. In *IJCAI*, volume 91, pages 331–337, 1991.

[152] Akinori Awazu and Kunihiko Kaneko. Ubiquitous glassy relaxation in catalytic reaction networks. *Physical Review E*, 80(4):041931, 2009.

[153] Jeremy M Berg, John L Tymoczko, and Lubert Stryer. Biochemistry, 2002.

[154] Jeffrey Van Ness, Lori K Van Ness, and David J Galas. Isothermal reactions for the amplification of oligonucleotides. *Proceedings of the National Academy of Sciences*, 100(8):4504–4509, 2003.

[155] Eric Tan, Barbara Erwin, Shale Dames, Tanya Ferguson, Megan Buechel, Bruce Irvine, Karl Voelkerding, and Angelika Niemz. Specific versus nonspecific isothermal dna amplification through thermophilic polymerase and nicking enzyme activities. *Biochemistry*, 47(38):9987–9999, 2008.

[156] H. Kong, J. Aliotta, and J.J. New England Biolabs Pelletier. *Bst* dna polymerase, large fragment.

[157] Daniel T Gillespie. Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 81(25):2340–2361, 1977.

[158] Thomas E Ouldridge, Rollo L Hoare, Ard A Louis, Jonathan PK Doye, Jonathan Bath, and Andrew J Turberfield. Optimizing dna nanotechnology through coarse-grained modeling: A two-footed dna walker. *ACS nano*, 7(3):2479–2490, 2013.

[159] Ibuki Kawamata, Fumiaki Tanaka, and Masami Hagiya. Abstraction of dna graph structures for efficient enumeration and simulation. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 800–806, 2011.

[160] Ibuki Kawamata, Nathanael Aubert, Masahiro Hamano, and Masami Hagiya. Abstraction of graph-based models of bio-molecular reaction systems for efficient simulation. In *Computational Methods in Systems Biology*, pages 187–206. Springer, 2012.

[161] Ishanu Chattopadhyay, Anna Kuchina, Gürol M Süel, and Hod Lipson. Inverse gillespie for inferring stochastic reaction mechanisms from intermittent samples. *Proceedings of the National Academy of Sciences*, 110(32):12990–12995, 2013.

[162] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

[163] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Computer aided verification*, pages 154–169. Springer, 2000.

[164] Ross D King, Jem Rowland, Stephen G Oliver, Michael Young, Wayne Aubrey, Emma Byrne, Maria Liakata, Magdalena Markham, Pinar Pir, Larisa N Soldatova, et al. The automation of science. *Science*, 324(5923):85–89, 2009.

[165] J. M. Smith. *Evolution and the Theory of Games*. Springer US, 1993.

[166] Barry Sinervo and Curt M Lively. The rock-paper-scissors game and the evolution of alternative male strategies. *Nature*, 380(6571):240–243, 1996.

[167] Benjamin Kerr, Margaret A Riley, Marcus W Feldman, and Brendan JM Bohannan. Local dispersal promotes biodiversity in a real-life game of rock–paper–scissors. *Nature*, 418(6894):171–174, 2002.

[168] Michael B Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.

[169] Tobias Reichenbach, Mauro Mobilia, and Erwin Frey. Mobility promotes and jeopardizes biodiversity in rock–paper–scissors games. *Nature*, 448(7157):1046–1049, 2007.

[170] Marcus Frean and Edward R Abraham. Rock–scissors–paper and the survival of the weakest. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 268(1474):1323–1327, 2001.

[171] Akio Namiki, Yoshiro Imai, Masatoshi Ishikawa, and Makoto Kaneko. Development of a high-speed multifingered hand system and its application to catching. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2666–2671. IEEE, 2003.

[172] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[173] Richard Cook, Geoffrey Bird, Gabriele Lünser, Steffen Huck, and Cecilia Heyes. Automatic imitation in a strategic context: players of rock–paper–scissors imitate opponents' gestures. *Proceedings of the Royal Society B: Biological Sciences*, 279(1729):780–786, 2012.

[174] Joanne Macdonald, Darko Stefanovic, and Milan N Stojanovic. Dna computers for work and play. *Scientific American*, 299(5):84–91, 2008.

[175] Agoston E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[176] Priscilla EM Purnick and Ron Weiss. The second wave of synthetic biology: from modules to systems. *Nature Reviews Molecular Cell Biology*, 10(6):410–422, 2009.

[177] Guillermo Rodrigo, Javier Carrera, and Alfonso Jaramillo. Computational design of synthetic regulatory networks from a genetic library to characterize the designability of dynamical behaviors. *Nucleic acids research*, 39(20):e138–e138, 2011.

[178] G Max Kelly and Ross Street. Review of the elements of 2-categories. In *Category seminar*, pages 75–103. Springer, 1974.

[179] Bryan Bergeron. Developing serious games (game development series). 2006.

[180] Salvatore Bommarito, Nicolas Peyret, and John SantaLucia Jr. Thermodynamic parameters for dna sequences with dangling ends. *Nucleic Acids Research*, 28(9):1929–1934, 2000.

Appendix A

Model parameters and equations

A.1 Coaxial-stacking slowdown

The slowdown due to coaxial stacking is computed following a simple two-state model: when the bases at the nick are stacked, the DNA complex is considered stable and, as such, does not denature; when the bases are not stacked, the release happens without at the normal speed (actually, faster, because there is no dangle stabilization). The reactions going from one state to the other are considered at equilibrium, which allows us to use the equilibrium constants as the slowdown ratio. Coaxial-stacking stability values were taken from Frank-Kamenetskii's group measures [106]. We used their formula on temperature and salt dependence [105] to correct those values for the conditions of Padirac *et al.* [11] (temperature 42°, salt equivalent of 120 mM of Na^+). Since we only care about the relative increase in stability compared to a single-stranded input or output present on the template, we have to reduce those values to take into account the stability increase of dangles [180] (values were corrected for the temperature, and averaged in the same way that Frank-Kamenetskii's group [106] to be able to compare). We then get:

Sequence at the nick (5′ to 3′)	Coaxial-stacking stability increase (kcal/mol)	Final stability increase	Equivalent slowdown factor
TA	- 0.46	0.09	1.2
CA = TG	- 0.85	-0.30	0.62
CG	- 1.21	-1.06	0.19
AG = CT	- 1.36	-1.14	0.16
AA = TT	- 1.41	-1.14	0.16
AT	-1.64	-1.33	0.12
GA = TC	- 1.73	-1.15	0.16
GG = CC	- 1.74	-1.34	0.12
GT = AC	- 2.11	-2.00	0.04
GC	-2.47	-2.03	0.04
Average	- 1.51	- 1.15	0.16

However, those values do not take into account the base pairs neighbouring the nicking site, although a strong dependence has been observed by Pyshnyi and Ivanova [104].

A.2 Full equation set for an autocatalytic module

In the case of a simple autocatalytic module $s \to s$, we get the following equations:

$$\begin{aligned}
\frac{\mathrm{d}[s]}{\mathrm{d}t}(t) \quad &= \quad k_{duplex} \cdot K_s \cdot ([temp_{in}](t) + [temp_{out}](t) + 2\,stack \cdot [temp_{both}](t)) \\
&- k_{duplex} \cdot [s](t) \cdot (2\,[temp_{alone}](t) + [temp_{in}](t) + [temp_{out}](t)) \\
&+ pol(t) \cdot [temp_{both}](t) - \; exo_s(t) \cdot [s](t)
\end{aligned}$$

$$\begin{aligned}
\frac{\mathrm{d}[temp_{alone}]}{\mathrm{d}t}(t) \quad &= \quad k_{duplex} \cdot ([temp_{in}](t) + [temp_{out}](t)) \\
&- 2\,k_{duplex} \cdot [s](t) \cdot [temp_{alone}](t)
\end{aligned}$$

$$\begin{aligned}
\frac{\mathrm{d}[temp_{in}]}{\mathrm{d}t}(t) &= k_{duplex} \cdot ([s](t) \cdot [temp_{alone}](t) + stack \cdot K_s \cdot [temp_{both}](t)) \\
&\quad - k_{duplex} \cdot [temp_{in}](t) \cdot ([s](t) + K_s) - pol \cdot [temp_{in}](t)
\end{aligned}$$

$$\begin{aligned}
\frac{\mathrm{d}[temp_{out}]}{\mathrm{d}t}(t) &= k_{duplex} \cdot ([s](t) \cdot [temp_{alone}](t) + stack \cdot K_s \cdot [temp_{both}](t)) \\
&\quad - k_{duplex} \cdot [temp_{out}](t) \cdot ([s](t) + K_s)
\end{aligned}$$

$$\begin{aligned}
\frac{\mathrm{d}[temp_{both}]}{\mathrm{d}t}(t) &= k_{duplex} \cdot [s](t) \cdot ([temp_{in}](t) + [temp_{out}](t)) \\
&\quad - 2\, stack \cdot k_{duplex} \cdot K_s \cdot [temp_{both}](t) \\
&\quad + nick \cdot [temp_{ext}](t) - pol \cdot [temp_{both}](t)
\end{aligned}$$

$$\begin{aligned}
\frac{\mathrm{d}[temp_{ext}]}{\mathrm{d}t}(t) &= pol \cdot ([temp_{in}](t) + [temp_{out}](t)) \\
&\quad - nick \cdot [temp_{ext}](t)
\end{aligned}$$

$$\begin{aligned}
exo_s(t) &= \frac{V_{m,exo}}{K_{m,short} \cdot \left(1 + \frac{[s](t)}{K_{m,short}} + \frac{[temp_{alone}](t)}{K_{m,temp}}\right)} \\
pol(t) &= \frac{V_{m,pol}}{K_{m,pol} \cdot \left(1 + \frac{[temp_{in}](t)}{K_{m,pol}} + \frac{[temp_{both}]}{K_{m,displ}}\right)} \\
nick &= \frac{V_{m,nick}}{K_{m,nick} + [temp_{ext}](t)}
\end{aligned}$$

A.3 Proof of equation 2.10

We place ourselves in the case of an autocatalytic module at equilibrium. Then it follows that all derivatives are equal to zero. The complete equation of the system are given above. Note that in the following, we will write the concentration of species at equilibrium without the $_{eq}$ notation to improve readability.

First, by reorganizing the terms in the derivative of s, we get:

$$\frac{\mathrm{d}[s]}{\mathrm{d}t} = \frac{\mathrm{d}[temp_{alone}]}{\mathrm{d}t} - \frac{\mathrm{d}[temp_{both}]}{\mathrm{d}t} + nick \cdot [temp_{ext}] - exo \cdot [s] = 0$$

It then immediately follows that

$$exo \cdot [s] = nick \cdot [temp_{ext}] \tag{A.1}$$

Moreover, we have

$$\frac{\mathrm{d}[temp_{ext}]}{\mathrm{d}t} = pol \cdot ([temp_{in}] + [temp_{out}]) - nick \cdot [temp_{ext}] = 0$$

This yields the result that is used with Equation A.1 to determine the fluorescence of Eva-Green in Chapter 2:

$$pol \cdot ([temp_{in}] + [temp_{out}]) = nick \cdot [temp_{ext}] \tag{A.2}$$

This also gives us an expression of the sum of three fifth of the internal states of *temp* as a function of $[s]$. We use combinations of the equations to determine the remaining two, first as an expression of $[temp_{in}]$:

$$\frac{\mathrm{d}[temp_{in}]}{\mathrm{d}t} - \frac{\mathrm{d}[temp_{out}]}{\mathrm{d}t} = k_{dup} \cdot (K_s + [s]) \cdot ([temp_{out}] - [temp_{in}]) - pol \cdot [temp_{in}] = 0$$

$$\Rightarrow [temp_{out}] = (1 + dfracpolk_{dup} \cdot (K_s + [s])) \cdot [temp_{in}] \tag{A.3}$$

Then, we have, from the derivative of $[temp_{alone}]$:

$$2\,k_{duplex} \cdot [s](t) \cdot [temp_{alone}](t) = k_{duplex} \cdot ([temp_{in}](t) + [temp_{out}](t))$$

Using Equation A.3, we get:

$$[temp_{alone}] \;=\; \frac{K_s}{2[s]} \cdot (2 \,+\, dfracpolk_{dup} \cdot (K_s + [s])) \cdot [temp_{in}] \tag{A.4}$$

From the derivative of $[temp_{in}]$, we get the following equality:

$$k_{duplex} \cdot [s] \cdot [temp_{alone}] \;=\; (k_{duplex} \cdot [s] \,+\, k_{duplex} \cdot K_s + pol) \cdot [temp_{in}] - k_{duplex} \cdot K_s \cdot coax \cdot [temp_{both}]$$

From here on, we will make the assumption that $coax = 1$. By using a rewriting trick ($k_{duplex} \cdot K_s \cdot [temp_{in}] - k_{duplex} \cdot K_s \cdot [temp_{in}] = 0$), we get:

$$\begin{aligned}
k_{duplex} \cdot [s] \cdot [temp_{alone}] \;&=\; (k_{duplex} \cdot [s] \,+\, 2k_{duplex} \cdot K_s + pol) \cdot [temp_{in}] \\
&\quad - k_{duplex} \cdot K_s \cdot ([temp_{in}] + [temp_{both}]) \\
&=\; (k_{duplex} \cdot [s] \,+\, 2k_{duplex} \cdot K_s + pol) \cdot [temp_{in}] \\
&\quad - k_{duplex} \cdot K_s \cdot \frac{exo}{pol} \cdot [s]
\end{aligned}$$

$$[temp_{alone}] \;=\; \left(1 \,+\, 2\frac{K_s}{[s]} \,+\, \frac{pol}{k_{duplex}[s]}\right) \cdot [temp_{in}] \,-\, K_s \cdot \frac{exo}{pol} \tag{A.5}$$

We then recognize an expression combining $[temp_{alone}]$ and $[temp_{out}]$, namely:

$$\begin{aligned}
2[temp_{alone}] + [temp_{out}] \;&=\; \left(1 \,+\, 2\frac{K_s}{[s]} \,+\, \frac{pol \cdot K_s}{k_{duplex}(K_s + [s])[s]} \,+\, \frac{pol}{k_{duplex}(K_s + [s])}\right) \cdot [temp_{in}] \\
&=\; \left(1 \,+\, 2\frac{K_s}{[s]} \,+\, \frac{pol}{k_{duplex}[s]}\right) \cdot [temp_{in}]
\end{aligned}$$

Injecting this result in Equation A.5, we finally have:

$$[temp_{alone}] \,+\, [temp_{out}] \;=\; K_s \cdot \frac{exo}{pol} \tag{A.6}$$

This yields the final result:

$$\begin{aligned}
[temp]_{total} \;&=\; [temp_{alone}] \,+\, [temp_{out}] \,+\, [temp_{in}] \,+\, [temp_{both}] \,+\, [temp_{ext}] \\
&=\; K_s \cdot \frac{exo}{pol} + exo \cdot \left(\frac{1}{pol} \,+\, \frac{1}{nick}\right) \cdot [s]
\end{aligned}$$

A.4 Reactions of the delay gate

In the following, we note O for output, T for timer, T_{ext} for extended timer, G for gate and F for fuel. When it is necessary to distinguish between the two possible conformations of the gate, we precise whether its $5'$ end is single-stranded (G_{up}) or double-stranded (G_{down}). Polymerase activity is represented by irreversible reactions (simple arrows).

Step 1:

$$(O.T) + G \underset{\text{slow}}{\overset{\text{fast}}{\rightleftharpoons}} (O.T).G \tag{A.7}$$

$$T + G \underset{\text{very slow}}{\overset{\text{very fast}}{\rightleftharpoons}} T.G \tag{A.8}$$

$$T + (O.T).G \underset{\text{very slow}}{\overset{\text{fast}}{\rightleftharpoons}} (O.T) + T.G \tag{A.9}$$

Step 2:

$$T.G \overset{\text{fast}}{\rightarrow} T_{ext}.G_{up} \tag{A.10}$$

$$(O.T).G \overset{\text{fast}}{\rightarrow} (O.T_{ext}).G_{up} \tag{A.11}$$

$$T_{ext}.G_{up} + F \underset{\text{fast}}{\overset{\text{fast}}{\rightleftharpoons}} T_{ext}.G_{up}.F \tag{A.12}$$

$$(O.T_{ext}).G_{up} + F \underset{\text{fast}}{\overset{\text{fast}}{\rightleftharpoons}} (O.T_{ext}).G_{up}.F \tag{A.13}$$

Step 3:

$$T_{ext}.G_{up} \underset{\text{fast}}{\overset{\text{fast}}{\rightleftharpoons}} T_{ext}.G_{down} \tag{A.14}$$

$$(O.T_{ext}).G_{up} \underset{\text{fast}}{\overset{\text{fast}}{\rightleftharpoons}} (O.T_{ext}).G_{down} \tag{A.15}$$

$$T_{ext}.G_{down} + F \underset{\text{fast}}{\overset{\text{fast}}{\rightleftharpoons}} (T_{ext}.F).G_{down} \tag{A.16}$$

$$(O.T_{ext}).G_{down} + F \underset{\text{fast}}{\overset{\text{fast}}{\rightleftharpoons}} (O.T_{ext}.F).G_{down} \tag{A.17}$$

Step 4:

$$(T_{ext}.F).G_{down} \overset{\text{fast}}{\rightarrow} W + G \tag{A.18}$$

$$(O.T_{ext}.F).G_{down} \overset{\text{fast}}{\rightarrow} O + W + G \tag{A.19}$$

Leak/capture reactions:

$$O + T \underset{\text{very slow}}{\overset{\text{very fast}}{\rightleftharpoons}} O.T \tag{A.20}$$

$$O + T.G \underset{\text{very slow}}{\overset{\text{very fast}}{\rightleftharpoons}} (O.T).G \tag{A.21}$$

$$O + T_{ext}.G_{up} \underset{\text{very slow}}{\overset{\text{very fast}}{\rightleftharpoons}} (O.T_{ext}).G_{up} \tag{A.22}$$

$$O + T_{ext}.G_{up}.F \underset{\text{very slow}}{\overset{\text{very fast}}{\rightleftharpoons}} (O.T_{ext}).G_{up}.F \tag{A.23}$$

$$O + T_{ext}.G_{down} \underset{\text{very slow}}{\overset{\text{very fast}}{\rightleftharpoons}} (O.T_{ext}).G_{down} \tag{A.24}$$

$$O + (T_{ext}.F).G_{down} \underset{\text{very slow}}{\overset{\text{very fast}}{\rightleftharpoons}} (O.T_{ext}.F).G_{down} \tag{A.25}$$

Reaction 3 represents the toehold mediated strand displacement of a duplexed timer strand attached to the gate by a single-stranded timer strand. Reactions 4 and 5 are polymerase extension of the timer strand. Similarly reactions 12 and 13 are polymerase extension of the fuel strand. Reaction 13 releases the output. Differential equations are directly derived from those reactions assuming mass action kinetics [99].

A.5 Sequences for the delay gate

Strand name	Strand sequence
Output 6	GCTTTCGTTCGCTTGCCGTCCGTGTC
Output 8	CTGCTTTCGTTCGCTTGCCGTCCGTGTC
Output 10	TTCTGCTTTCGTTCGCTTGCCGTCCGTGTC
Output 12	GGTTCTGCTTTCGTTCGCTTGCCGTCCGTGTC
Output 14	CTGGTTCTGCTTTCGTTCGCTTGCCGTCCGTGTC
Fuel-driven Gate, missing dNTP	TTGCGTGTC**AAAA**TGGCTTCCCA**TTTT**GA CACGCAACTCGGTCTGGTTCTGCTTTCTTT
Fuel-driven Gate, Sp-18 spacer	TTGCGTGTCAAATGGCTTCCCATTT-**spacer-** GACACGCAACTCGGTCTGGTTCTGCTTTCTTT
Fuel-less Gate, missing dNTP	GCGGCTTGTGCCGCTTGTC**AAAA**TGGCTTCCCA **TTTT**GACAAGCGGCACAAGCCGCTTGTCGGTCTGCTTT
Timer	GACACGGACGGCAAGCGAACGAAAGCAGAACCAGACCGAG
Fuel	GACACGCAA

Table A.1: Sequences of the different strands in the system, given from 5′ to 3′. The numbers in the name of the output sequences represent the length n of the part of the output overlapping with the gate toehold. Bold text in the gate sequence represents the blocker sequence and its complementary.

Appendix B

Tutorial DACCAD

B.1 Introduction

DACCAD (DNA Artificial Circuits Computer Assisted Design) is a Computer Assisted Design software for Montagne *et al.*'s experimental framework for the building of dissipative reaction networks, dubbed the DNA (Dynamic Networks Assembly) toolbox [67]. Its goal is to help both first comers and experienced users to create quickly DNA toolbox systems, either as a proof of concept or for an actual *in-vitro* implementation. The last section gives some details on how to match the model parameters to your experimental setting.

The DNA toolbox is a set of modules aimed to simplify the design of dynamic DNA computing systems. The basic idea is to keep the system out of equilibrium by continuously generating and degrading short DNA strands. Those strands play a similar role to signaling compounds in biological regulation networks, or to variables in a computer program. They interact with specific longer strands ("templates") which are kept stable over time and can be seen as the "program" of the system.

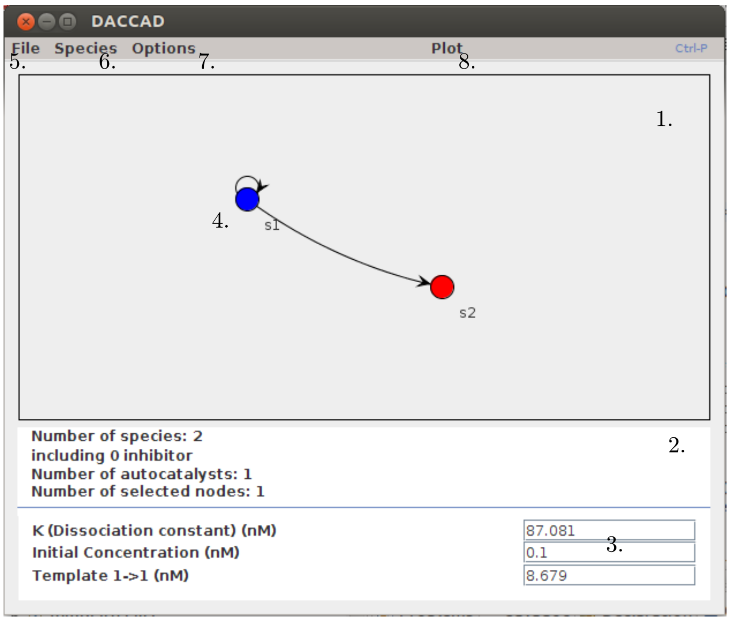
Specifically, the toolbox is composed of three modules: activation, autocatalysis (which can be considered as a special case of activation) and inhibition. Generation is assured by polymerization and nicking, degradation by the action of exonuclease. DACCAD leverages the simple graph representation of DNA toolbox systems to allow the user to easily create and manipulate those systems while automating the ODE generation.

Enzymatic activities and affinities can be set in the options. The default settings are from Padirac *et al.* [11]. See the supplementary material of [67] to experimentaly find values appropriate to your own experimental settings.

B.2 Interface

B.2.1 Main window

This window is used to display the DNA system being designed as well as general information relative to it.



Number of species: 2
including 0 inhibitor
Number of autocatalysts: 1
Number of selected nodes: 1

K (Dissociation constant) (nM) 87.081
Initial Concentration (nM) 0.1
Template 1->1 (nM) 8.679

1. Display panel. Shows the graphical representation of the system.

2. Data panel. Shows general data about the system (top) as well as information specific to the selected node(s) (bottom).

3. Parameters area. Allows the user to modify values related to the selected node(s), such as DNA sequence stability or DNA templates concentration.

4. Graph of the system. Nodes represent DNA signal species and arrows DNA templates. Selected nodes are displayed in blue.

5. File menu.

6. Species menu. Groups species related operations.

7. Options. Allows the user to change parameters about the simulator, the enzymes and optimization.

8. Plot. Simulates the system and display the result in a new window.

The display panel represents the system as a graph, following the conventions of section 3. The drawing itself is done by the JUNG library [137]. It features the following possibilities:

- Species (nodes) can be freely added and removed to the system. Activation templates are simply added by first selecting the signal species and then the output species. Inhibitions are added through an option at species creation, allowing the new species to be the inhibitor of an existing module.

- Nodes can be freely moved around by drag and drop. If more than one node is selected, then their relative positions are kept. This change is purely aesthetic and does not change the system itself.

- Additional structures can be added to the current system either by selecting multiple nodes and using the "duplicate" feature, or by importing a previously saved system. In the duplication case, only templates joining selected nodes (as opposed to templates joining a selected node to an unselected one) will be duplicated.

- Selected nodes can be fused together, with a best effort policy to maintain the coherence of the structure. If a module use one of the selected nodes before fusion, it will be replaced by a similar module using the fused species instead. Duplicates are then removed. The fusion detects direct autocatalysis, in which case an autocatalytic template is added to the new node. Inhibition species are simply removed. This mechanism is useful to quickly combine multiple structures imported from an existing database.

- Right-clicking on a node displays a contextual menu with two options: deleting the node and managing the species inputs, that is the external injections of the species into the system. It provides two parameters: the time at which the injection is made, and the added concentration in case of a spike. Other kinds of inputs can be described in a separate file and imported. Inputs can be periodic, in which case the period can also be set.

The data panel displays the following information:

- General data on the system: the total number of species, which can be seen as a degree of complexity of the system, the number of inhibiting species, the number of direct autocatalysts and the number of currently selected nodes.

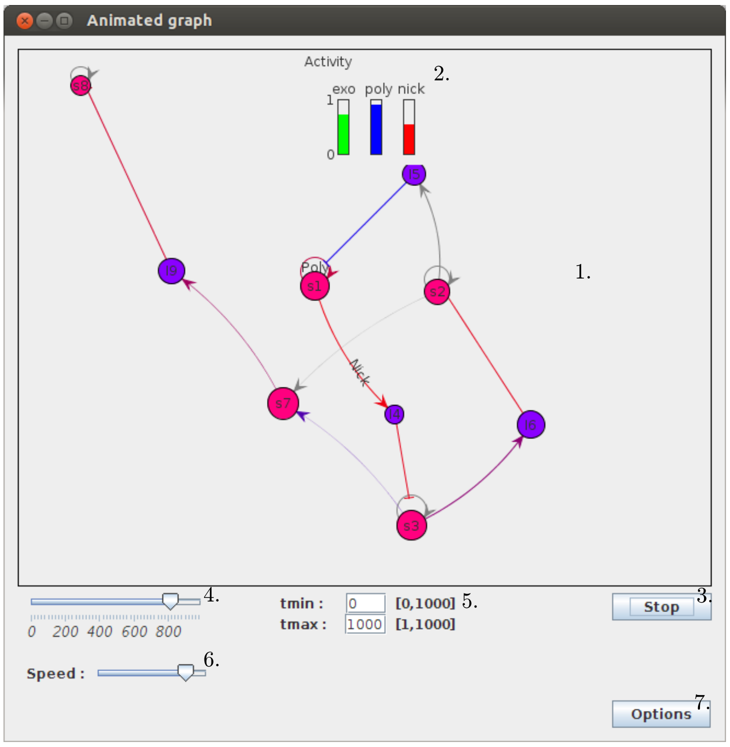
- Contextual data, depending on the selected nodes. If one species is selected, its initial concentration and K_s (dissociation constant) are displayed and can be adjusted. If this species has an autocatalytic module, the concentration of template of this module can also be set. If two species are selected the concentration of template of modules using one as signal and the other at output are displayed, with a maximum of two if both species are activating each other. The edition area also contains an button opening a window with additional options, such as deleting the template or setting its coaxial slowdown (default 0.2).

The option menu allows to set all relevant parameters to simulate the system, notably the activity and Michaelis parameters of all enzymes along with the possibility to declare them saturable and to toggle competitive coupling, the K_{duplex}, α and simulation time. It is also possible to declare which species concentration should be displayed in the time plot and set which parameters can be modified by the optimiser.

Finally the plot button commands the simulation of the system using the Gragg-Bulirsch-Stoer integrator from the Apache Commons Mathematics Library. The resulting time trace is then displayed using the JFreeChart library [138] in a separate window. It can also be exported as a text file to be used with other programs, such as Matlab or Gnuplot.

The remaining actions are standard ("new", "save", "open") or straightforward ("export").

B.2.2 The animate window



1. Display of the graph. Nodes radius: concentration of the species. Nodes color: stability as expressed by the dissociation constant, blue is stable (dissociation constant low), red unstable (dissociation constant high). Edges thickness: total template concentration.

Edges color: quantity of loaded or active template (that is, anything that is not free or inhibited), gray means less than 10% active, then the scale goes from blue to red. Nick, poly, exo: each mark represents which template is saturating said enzyme the most. If a given enzyme is saturated to the point of changing the behavior of the system, those marks can be used to find the bottleneck.

2. Saturation panel: shows the current observed activity of each enzyme. 1 means the enzyme is available at full speed (no substrate), 0 means the enzyme is completely saturated.
3. Animate/Stop button: starts (respectively stops) the animation.
4. Time: represents which moment of the simulation is displayed.
5. Minimum and maximum time: sets which part of the simulation should be animated.
6. Speed scale: sets the animation speed.
7. Options: allows to set some appearance options, such as whether to display the enzyme saturation marks or how to calculate the nodes radius.

B.3 DACCAD tutorials

This section gives four tutorials presenting the possibilities of DACCAD.

Note: if at any time you make a mistake or would like to go back, you can use the undo option from the “File” menu (shortcut Ctrl+Z). It is also possible to remove nodes with right click (secondary click on Mac OS)>remove. Another possibility is to select the nodes and press “Del”. Templates can be removed by selecting the nodes it connects and pressing the “X” button in the data panel.

Similarly, enzymatic parameters can be reset, using the “default” button in the “Enzymes” tab of the options. Default parameters for sequences dissociation constants are (roughly) 74 nM for a signal species and 0.2 nM for an inhibiting species. Templates have a default concentration of 10 nM, with 0.2 coaxial slowdown.

Tutorial 1. A simple logical gate: the NOT operation

This tutorial will highlight:

- the creation of signal species and templates.

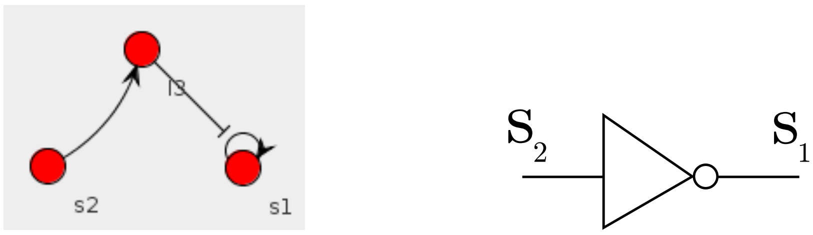


Figure B.1: Left: inverter gate made with the DNA toolbox. Right: logical diagram. s_2 is the input and s_1 the output.

- parameter setting.
- pulse inputs.

Logical circuit make great use of the NOT gate. Its working is simple: when it has an input, no output should be produced. Conversely, in the absence of input, there should be an output. In the DNA toolbox, an inverter (or NOT gate) is made of an autocatalyst that can be inhibited by the action of a second signal species. The diagram of the gate and its logical equivalent are shown in Figure B.1.

First step: create a new activation species. This can be done through the “New species” function in the “Species” menu (shortcut Ctrl+A). In the pop-up window, choose “activation species”. A new species named s_1 should appear. Select it by clicking on it (it will become blue) and change its initial concentration (last line of the data panel) to 1 nM.

Second step: Add the autocatalysis template. Adding a template creating a species b from a species a is simple: select a and then click on b. In the present case, s_1 should already be selected, so it is enough to click on it again. The graph should look like the one on Figure B.2, left. You can simulate this network by pressing “plot” (shortcup Ctrl+p), which should yield a nice steady state after transient amplification. If the curve is completely flat, check that the initial concentration of s_1 is not zero.

Third step: add a new activation species (Figure B.2, center). It is possible to reposition it by drag-and-drop.

Fourth step: add a new species, select inhibition. An inhibition species always targets a specific template, so the application will open a pop-up window to ask which template should be inhibited. Note that, due to restrictions in the DNA toolbox, only templates generating signal species can be inhibited [1]. In the present case, there should be only one choice (“1->1”), press “ok”. Add a template from s_2 to the inhibitor (Figure B.2, right). The NOT gate is now complete.

[1]To work around this limitation, it is possible to use an indirect activation. That is, a signal species activates another signal species which in turn creates the inhibitor. It is then possible to inhibit the first activation.

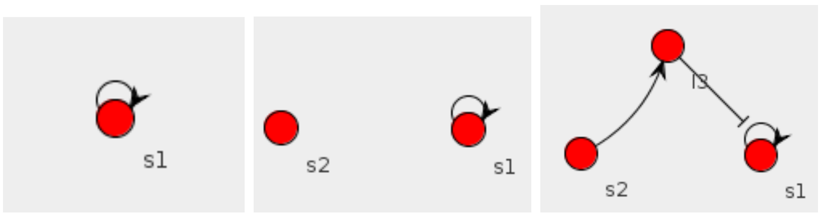
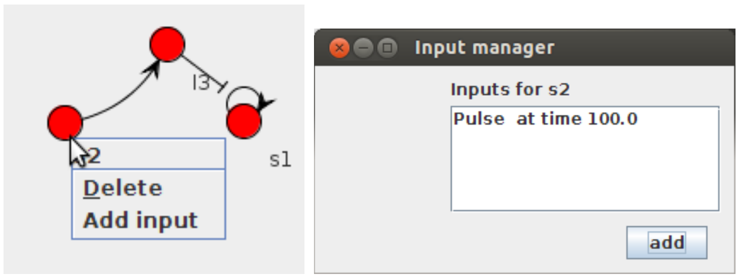


Figure B.2: NOT gate: step by step creation of the network. The output is shown in dark blue, inputs respectively in red and cyan.



Inputs for s2

Pulse at time 100.0

Figure B.3: Adding pulse inputs

To test the behavior of the NOT gate, we can add a pulse input on the species s_2. This input represents an injection of s_2 in the system.

To open the input manager, right click (secondary click on Mac OS) on the species s_2 and select “add input” (Figure B.3, left). This should open a new window. Just click add and close (Figure B.3, right). It is also possible to modify input parameters (such as input time, quantity or periodicity) or to delete an input by selecting an existing input in the input manager. Species can have any number of inputs, which are all managed through the input manager window. Species with inputs are displayed double-circled.

We can then use the plot option (shortcut Ctrl+p) to observe the simulated behavior of the system. We obtain indeed a signal inversion (Figure B.4). Note that it is possible to automatically resimulate and replot a system every time a change occurs by toggling “autoplot” on in the “Options” menu.

Finally, we can save this system in convenient place, using the “File>Save” command.

Tutorial 2. Creating a more complex gate: the AND gate

This tutorial assumes that the first tutorial was completed. It focuses on:

- multiple selection and multiple nodes reorganization.
- advanced graph manipulation: the “join” and “duplicate” operations.
- stability modifications.

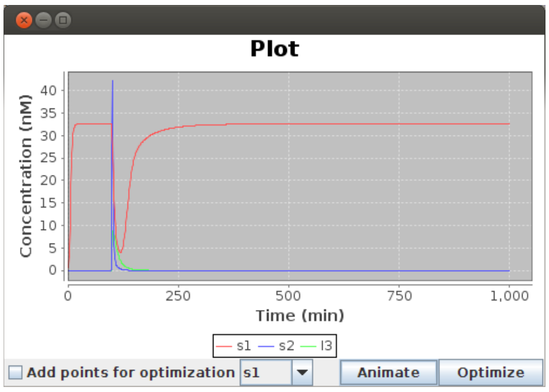


Figure B.4: Behavior of the NOT gate. When s_2 is injected at time 100, the output signal s_1 is drastically reduced, before returning to normal when s_2 is hydrolysed.

- continuous inputs.

In this tutorial, we will realize an AND gate, building on the previous NOT gate. We will then add continuous chemical inputs to the logical inputs of the circuit to observe its response to external signal. In a wet lab, such continuous input can be realized by using microfluidic devices [86].

First step: starting from the NOT gate, either by opening a previously saved file or after the previous tutorial, select all nodes. To do so, you can click on an empty space and drag the pointer across the screen to perform a multiselection, or use Shift + left click to toggle which nodes are selected. Once all nodes are selected, use the "Species>Duplicate selected". Drag and drop the selected nodes so that the network looks like Figure B.5, left.

Second step: Click on an empty spot to clear the selection. Select the output of the first gate and the input of the second gate (respectively s_1 and s_6 on Figure B.5, left). Use the "Species>Join selected" operation to fusion those two nodes. Reorganize the network as in Figure B.5, center. Set the initial concentration of all autocatalysts to 1 nM.

Third step: Simulate the system with the plot option. If there is no pulse at time 100, add a pulse input on the circuit input (s_2 here). We can see that the pulse is not strong enough to propagate to the output of the circuit (s_4). Indeed, the species gets degraded before it was able to have enough impact. To solve this problem, it is possible to use more stable species. Select s_2 and change its dissociation constant to 4 nM. The lower the dissociation constant, the longer it takes the species to detach from a template. After plotting again, we can see that there is now a large impact on the output.

Fourth step: We can also use this inertia to create an AND gate. This gate should have an output if and only if both its input are present. To do so, we will create a new activation species (see tutorial 1) and connect it to the first inhibitor (Figure B.5, right). Set the dissociation

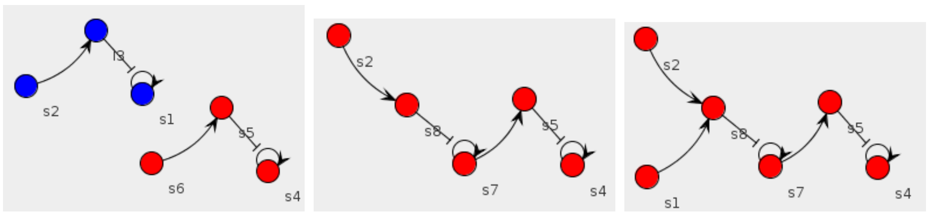


Figure B.5: Creation of the logical circuit.

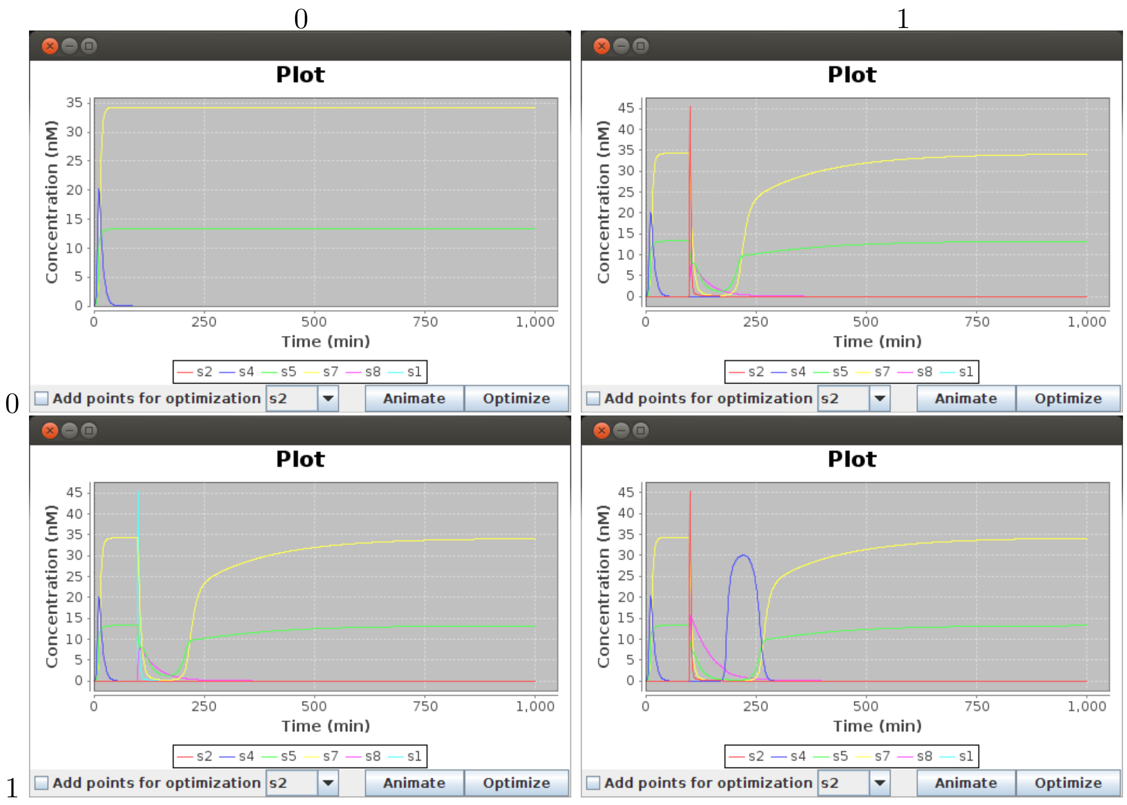


Figure B.6: Truth table of the AND gate.

constants of both species to 20 nM. Replot. You can see that there is a noticeable impact on the system, but no output is created. Add a pulse input to the new species at time 100. Replot. There is now a spike of output in the system after a delay. The whole set of possibilities is shown in Figure B.6 and is the same as a logical AND gate.

Fifth step: finally, we can add an external continuous input representing an flow of chemical species, like in an microfluidic open reactor, for instance. To do so, create a simple file in a convenient place. Edit it with a program that will not add formatting, such as Emacs or Windows Notepad. Each line of this file should contain a value representing the added quantity in nM at the n^{th} minute, where n is the line number. In our case, copy and paste zeros for approximately 40 lines. This will leave enough time to the circuit to reach its stable state. You can then add any value for the next couple of lines. Floating point and negative values are accepted. If the file is longer than the simulation time, extraneous values are ignored. If the file

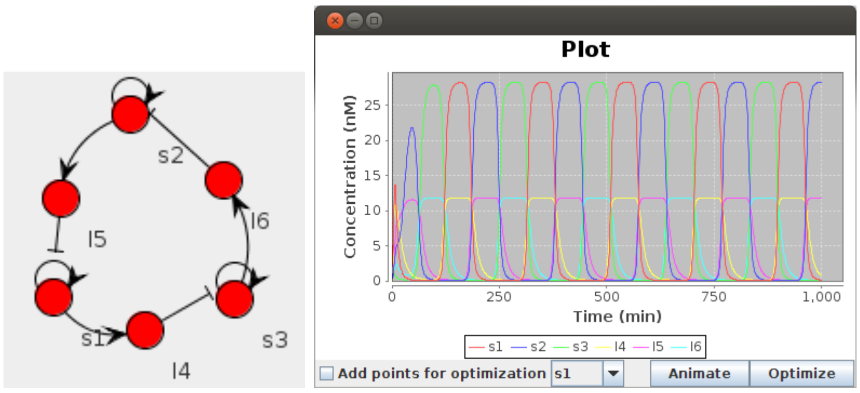


Figure B.7: Padirac's switch oscillator [43].

is too short, it is padded with zeros. Then return to DACCAD and choose "Species>Import input". Add the input to s_2 and replot. Species can have any number of inputs, which are all managed through the input manager window.

Tutorial 3. Importing modules and animation

This tutorial requires that you completed the first tutorial and saved the resulting system. This tutorial focuses on:

- setting templates concentrations.
- importing previously saved modules.
- the animation view of DACCAD.

In this tutorial, we will use an oscillating scheme to modulate a downstream module and give it a spiking behavior. Indeed, it is often necessary to create spikes, to switch bistable for instance. We will then observe and understand the resulting behavior with the visualization tool of DACCAD.

First step: This tutorial will create a large system, which is subject to enzymatic saturation. While it would be possible to make the current system robust against saturation, it is beyond the scope of the present tutorial. Moreover, when designing a system, it may be better to first turn off enzymatic saturations, check the integrity of the design, then toggle the saturations back on and optimize the parameters. To turn off the saturations, go to "Options>Enzymes" and turn all "saturable" checkboxes off. Now enzymatic activity is assumed to be first order.

Second step: Follow the pattern show in Figure B.7, left. Start all the autocatalysts by setting the initial concentration of s_1 to 5 nM, s_2 and s_3 to 1 nM. Plot. You should obtain the results shown in Figure B.7, right.

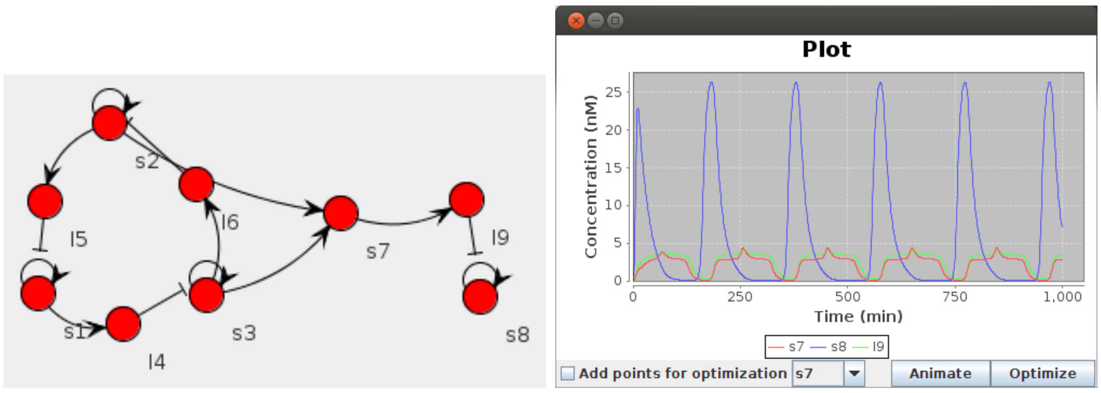


Figure B.8: The spike generator.

Third step: Use "Species>Import module" to import the previous NOT gate. The import operation allows you to add a previously saved system without overriding any parameter and is useful to quickly assemble a large system made of basic modules. Move the added gate to a convenient spot on the screen.

Fourth step: Connect both s_2 and s_3 to the input of the gate (Figure B.8, left).

Fifth step: The current system generates too much inhibitor, which destroys the output. Select both s_2 and the input of the gate (s_7 here). You can do so by using Shift + left click to add a species to the current selection. Then, set the template concentration to 1 nM. Do the same with s_3 and s_7. Also set the template from s_7 to the inhibitor of the NOT gate to 5 nM. You should observe the result shown in Figure B.8, right. You can also try different combinations of the template concentrations to change the shape and intensity of the spikes.

Sixth step: Toggle the saturations back on. You will notice that the shape of the plot is not optimal anymore, with large spikes. One way to change this is to play with the coaxial stacking slowdown of the templates. This effect represents the capacity of two DNA strands to stabilize each other when they are only separated by a nick. While not as stable than a full strand of the same length, the strands still denature slower than when alone on the template. This slowdown is sequence (and thus template) dependent, and can be access through the options of a given template (button on the right of the template concentration in the data panel). Based on multiple sources [98, 104], realistic values seems to be between 1.0 (no slowdown) and 0.04 (25 times slowdown).

Once you are satisfied with the plot, you can observe the dynamical evolution of the system by pressing "Animate". In the new window, press "Animate" to start the animation. You can get a better visualization for this system by opening the options, setting the radius scale to "linear" and the radius size factor to 0.3, for instance.

Tutorial 4. Optimization

This tutorial is independent of the previous ones and explains how to use the optimization tool. It is recommended to have done the first tutorial to know the basics about graph creation and edition.

We will realize a simple system with an autocatalyst activating an output, then use the optimization tool to change their respective levels.

First step: Create the system. Add an autocatalyst, set its initial concentration to 1 nM, add an activation species and finally add a template from the autocatalyst to this new species. The resulting graph is shown in Figure B.9, top.

Second step: When plotting, you can see that both steady-states are identical and that both curves are similar. We can use CMA-ES [92] to set a different level for both species. However, we don't want to evolve all possible parameters, we only want to change the template concentrations. Go to "Options>Optimization", select "Species dissociation constants" and uncheck the "optimize" option. Do the same with "Initial species concentrations". Since we are evolving two different species at the same time, it is a good idea to get a stricter (that is lower) threshold ratio. 0.5 should be a good starting point. The threshold ratio defines how close to the user-defined points the plot has to be on average. Sigma represents the initial standard deviation used by CMA-ES to modify the evolved parameters. If the fitness (that is the distance to the target) does not change much with respect to a given parameter, the algorithm will increase this parameter's sigma. Conversely, if the fitness changes fast, the algorithm will reduce the parameter's sigma.

Third step: Close the options. If you closed the plot window, plot again. Our goal is to get 40 nM of output for 20 nM of input. To change the display range, right-click on the plot, then go to "Properties>Plot>Range Axis>Other>Range". Uncheck the "Auto-adjust range" box and select a maximum of 45. Then close the properties window, check the "Add points for optimization" box and add two point for s_1 at the 20 nM level. Then use the drop box to select s_2 and add two points at the 40 nM level. The resulting screen should look like Figure B.9, bottom left. If you added a point by mistake, you can always remove it by clicking on it. You can also add points by loading a previously saved optimization file. The loaded points will be added to the target of the species currently selected.

Fourth step: Press "Optimize". After a short while, the species steady state should be correct (Figure B.9, bottom right). You can cancel the optimization at any time by closing the optimization window. The optimization will then stop at the end of the current generation. You

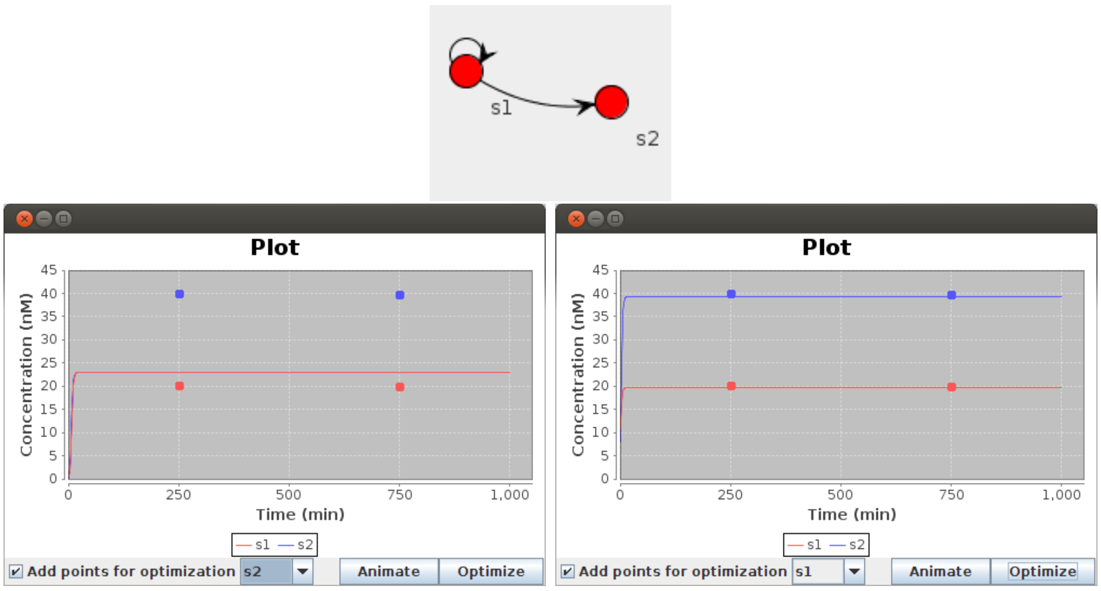


Figure B.9: Optimization of parameters.

can then select either s_1 alone or both s_1 and s_2 to check the updated templates concentrations.

B.4 File formats

This section describes the multiple file formats. While not explicitly human readable, they can be edited by hand, or generated by another resource.

B.4.1 Graph files

Graph files should have the ".graph" extension and contains three mandatory sections and two optional ones. It is possible to include commentaries, as the parser will ignore any line starting with a sharp symbol (#). The sections are as follow, and should appear in this order:

- SEQ: describes the species that are present in the graph. A line in this section contains the following data, separated by a tabulation: species ID, species initial concentration, node x coordinate in the layout, node y coordinate, sequence stability.
- TEM: the templates of the graph. The values represent the name (purely aesthetic, the name displayed in the data panel), input species ID, output species ID and total concentration.
- INHIB: which species are inhibitors, and of which template. Inhibiting species ID, template name (should be the same as in the previous section), template input species, template output species.

- INPUTS: optional, describes species inputs. Species ID, type (“pulse” or “file”), type specific data (time and concentration for a single pulse as well as frequency for a periodic pulse, URL for a file).
- PARAMS: optional, gives the system parameters. Parameter name, value. Each line is optional, and can be given in any order. The parameters abbreviations are:
  - absprec: absolute precision for integration.
  - relprec: relative precision for integration.
  - inhfact: stability ratio between an inhibitor on its template (fully double-stranded) and on the template it inhibits (double-stranded with mismatches). The higher this value, the less stable the inhibitor when inhibiting. This value cannot physically be inferior to 1, but this is not prevented by DACCAD.
  - diplrat: polymerase activity slowdown when displacing the output of a template. Only used for long outputs (i. e. inhibitors).
  - exokmib: exonuclease Michaelis-Menten parameter (K_m) with respect to inhibiting species.
  - exokmsi: exonuclease Michaelis-Menten parameter (K_m) with respect to signal species.
  - exokmtm: exonuclease Michaelis-Menten parameter (K_m) with respect to free templates. Even though they are not degraded, they competitive inhibitor of the exonuclease.
  - exovm: exonuclease reaction speed. Enzymatic activity is defined as $\frac{V_m}{K_m}$ with the relevant K_m.
  - kduplex: hybridization speed of complementary single strands.
  - nickkm: nicking enzyme Michaelis-Menten parameter (K_m).
  - nickvm: nicking enzyme reaction speed. Enzymatic activity is defined as $\frac{V_m}{K_m}$.
  - maxtime: simulation time in minutes.
  - polkm: polymerase Michaelis-Menten parameter (K_m) with respect to templates without output.
  - polkmbo: polymerase Michaelis-Menten parameter (K_m) with respect to templates with both input and output.
  - polvm: polymerase reaction speed. Enzymatic activity is defined as $\frac{V_m}{K_m}$ with the relevant K_m.

  - selfsta: polymerase self-start ratio. Represents the activity slowdown when the polymerase extends a template without primer.
  - toehold: hybridization slowdown when a template input or output displaces an inhibitor.

Here is a minimal working example:

```
SEQ
1 0.0 44.446860742124116 95.64789361292605 74.32767762106766
TEM
1->1 1 1 10.0
INHIB
```

B.4.2 Input files

Input files are simple text files with a value per line. The n^{th} line represents the flux (in nM.min^{-1}) at the n^{th} minute, values for the flux in between are computed using quadratic interpolation. Note that negative values are acceptable. If the file is not long enough for a given simulation (less lines than simulation minutes), it will be padded with zeros.

B.4.3 Optimization files

Those files are used to store optimization profiles. Each line represents an optimization point, composed of two values separated by a tabulation character. The first value represent the targeted time, the second the targeted concentration. Note that this file does not specify which species is the target, leaving it as a choice for the user. If multiple species should be optimized at the same, the various profiles have to be stored in different files.

B.4.4 Exported files

There are three types of exported files:

- raw simulation data: each line represents the concentrations of all species in the system at a given time (one minute per line), separated by a tabulation character.
- SBML file: exports the whole system as an SBML file, which can be used with other applications, such as Copasi.

- Mathematica file: exports a slightly simplified version of the system as a Mathematica file. Some options are not available yet in this format (inhibitor displacement and polymerase self-start), but will be added soon.