Computer-assisted design for scaling up systems based on DNA reaction networks

Nathanaël Aubert¹, Clément Mosca², Teruo Fujii², Masami Hagiya¹ and Yannick Rondelez²

¹Graduate School of Information Science and Technology, and ²LIMMS/CNRS-IIS, Institute of Industrial Science, University of Tokyo, Tokyo, Japan

In the past few years, there have been many exciting advances in the field of molecular programming, reaching a point where implementation of non-trivial systems, such as neural networks or switchable bistable networks, is a reality. Such systems require nonlinearity, be it through signal amplification, digitalization 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 counterintuitive process. Moreover, realistic prediction of the actual evolution of concentrations over time requires a number of side reactions, such as leaks, cross-talks or competitive interactions, to be taken into account. In this case, the design of a system targeting a given function takes much trial and error before the correct architecture can be found. To speed up this process, we have created DNA Artificial Circuits Computer-Assisted Design (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. We illustrate its effectiveness by designing various systems, from Montagne et al.’s Oligator or Padirac et al.’s bistable system to new and complex networks, including a two-bit counter or a frequency divider as well as an example of very large system encoding the game Mastermind. In the process, 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. 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 optimization algorithm that will automatically evolve parameters chosen by the user to try to match a specified behaviour. Finally, because 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. DACCAD can be downloaded online at http://www.yannick-rondelez.com/downloads/.

1. Introduction

1.1. Generality

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 promise [1–4], 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 towards actual applications. In this study, we present DNA Artificial Circuits Computer-Assisted design (DACCAD), a CAD program for designing systems using Montagne et al.'s [3] dynamic networks assembly (DNA) toolbox, a set of DNA-and-enzyme-based modules that can be arbitrarily combined into chemical reaction networks (CRNs).

CRNs are assemblies of cross-interacting chemical reactions where each molecular compound can affect the formation and degradation rate of the others. CRNs, 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 CRNs in vitro [5]. The recent demonstrations that they possess Turing universality [1,6–9] also advocates for their use in a variety of control tasks at the molecular level. However, CRNs tend to resist rational design. A link between the properties of the network and its dynamic functions can be obtained from dynamical systems theory, including linear stability analysis [10], Lyapunov stability analysis [11] or CRN theory [12], but such analytical approaches are generally restricted to idealized systems with simple mathematical models. Actual implementations are generally much more intricate in terms of chemical kinetics.

In recent years, great efforts have been invested in the exploration of the class 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 [6,13,14] but also neural networks [2] and CRNs [7]. Enzymatic systems are of particular interest because they can be maintained out of equilibrium for an extended period of time (for instance by having a large amount of deoxyribonucleotide triphosphate or nucleoside 5’-triphosphate in the solution, acting as a generic fuel) and thus are suited to explore emergent behaviours. We focus on the DNA toolbox, a recent framework to assemble out-of-equilibrium networks of arbitrary topology, whose dynamics are powered by three enzymes (polymerase, exonuclease and nickase). This framework has been used to build oscillators [3], bistable systems [4], a push–push memory circuit [4] or even molecular ecosystems reproducing predator–prey cycles in bulk [15], spatially [16,17] or in droplets [18]. Spatial implementations of those systems display waves [16,17], showing they can be used for two-dimensional reaction–diffusion systems. All these circuits 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 solutions, 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 counterintuitive effects, such as competition and saturation [19–21], have to be taken into account.

However, a quite unique feature of DNA as a molecular material is the possibility to find simple rules and kinetic parameters describing its global behaviour [7,22,23]. The dependence of each nucleotide on the global environmental conditions (salt concentration, temperature) is thus averaged through those parameters. While those rules are simple in essence, complexity emerges from their intrinsic nonlinearity. Similarly, the biochemistry of DNA can be described by simple models, such as the Michaelis–Menten equation or competitive inhibition. Therefore, DNA-based CRNs can be both implemented in test tubes and described by quantitative or semi-quantitative ordinary differential equation (ODE) models based on standard kinetics. Compared with previous experimental studies on CRNs, the availability of such mathematical models is the true benefit of DNA systems. It puts the molecular programming approach in stark contrast to ‘classic’ nonlinear reaction networks, such as those observed in chemistry (such as the Belousov–Zhabotinsky reaction [24]) and in biology (gene regulation networks [25,26], signalling cascades in metabolic networks [27], etc.). Note that in these last two cases, it is generally possible to infer high-level models of interaction among the molecular members of the reaction network, such as Hill equation for the cooperative binding of ligands [28]. However, it is much more difficult to obtain precise mathematical forms describing these interactions, let alone accurate numerical parameters needed for predictive simulations. This last point thus makes those systems harder to use in an engineering context.

DNA-based systems thus provide the chemical tools to implement any computation or dynamic behaviour at the molecular scale. While the construction of complex systems quickly becomes intractable for the human brain, these accurate mechanistic models allow us to numerically simulate them with reasonable accuracy, justifying CAD. 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 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 optimization tasks such as adjusting the parameters to tune the behaviour of the network towards a quantitative target, once a qualitative agreement has been obtained; and
— it can display an animated version of the network, giving a better understanding of the evolution of the system over time.

DACCAD was written entirely in Java1 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. Users can also refer to the electronic supplementary material of Montagne et al. [3] to fit these parameters to their own experimental settings. It comes with preset parameters from Padiarac et al.’s work, so that only a general knowledge of CRNs is necessary to start designing complex systems. DACCAD thus makes it quick to test designs based on the user’s intuition. Designed systems can then be built in vitro, using Baccouche et al.’s [29] protocol. Finally, DACCAD can convert any such designed system to synthetic biology markup language (SBML) to ensure compatibility with other simulation tools that the user would like to use.
1.2. 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 paradigm, DNA strand displacement (DSD), already benefits from its own software for CAD, VISUALDSD [30], demonstrating the effectiveness of the CAD approach. Recently, Kwiatkowska and co-workers [31] were able to prove the incorrectness of a basic DSD system by coupling VISUALDSD with a verification software and then propose a corrected version. VISUALDSD is also useful to highlight counterintuitive phenomena such as the winner-take-all effect [19,32,33].

DSD is also the foundation of Qian and Winfree’s seesaw gate [1,2], 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 hundreds of DNA sequences making up the system. This is why they designed a compiler (http://www.dna.caltech.edu/SeesawCompiler/) which takes a description of a logical circuit and outputs the DNA sequences to implement it. The compiler can also export the system in the MATHEMATICA and SBML [34,35] format for simulation of the system at the molecular level, and VISUALDSD format for simulation at the toehold level.

The motivation to automatically design DNA sequences is that unwanted interactions at the molecular level are very common. While simulation might show no problem with abstract sequences, actual implementation, if done incorrectly, can lead to unexpected secondary structures. In particular, because sequences are made up of only four nucleotides, it is hard to prevent partial complementarity. NUPACK [36] and DINAMelt [37,38] are online 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 kinds of DNA strands, whereas DINAMelt provides various data on the interaction of two sequences.

For broader applications, COPASI [39] is a very complete tool for analysing and simulating 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 optimization, which makes it very powerful. However, it has the inconvenience of requiring the user to provide the SBML file describing the system, or input each and every single molecules, interactions and reaction rates by hand through the graphical user interface, a long process for realistic systems. For this reason, 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. Because 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 [40], or by using directly specific short strands [41,42]. Such approaches have to rely heavily on CAD software, such as CADNA [43] and CanDo [44], as the number of different sequences necessary to create complex structures is very large.

Note that those programs 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.

1.3. The dynamic networks assembly toolbox

The DNA toolbox was introduced by Montagne et al. [3] to help rationally design DNA-based molecular programs, and demonstrated its effectiveness through multiple systems such as the Oligator. It has the advantage of using very simple modules (activation and inhibition) which can be combined to form, in theory, arbitrarily complex systems. Moreover, it was described both from the theoretical [45] and experimental point of view [29]. In the toolbox, two kinds of DNA strands have to be distinguished: short DNA strands are used as signals, and longer strands are used as templates to generate new strands (figure 1). Activation is done by short strands being extended by a DNA polymerase enzyme along a compatible template, after which both signal and output strands are cleaved apart by nickase. Both signal and output are too short to form stable duplexes and are eventually released. Inhibition is done by slightly longer strands that are complementary to most of their target, but do not trigger polymerization or nicking, thus inactivating the template. 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.

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 from their signal to their output. A second kind of arrow (‘bar-headed’ arrow) is also used to represent which template is inhibited by a given inhibition strand. Simple examples of such representation are shown in figure 2. Conversely, any such graph can be directly converted to a DNA implementation.

DACCAD can simulate both autonomous and non-autonomous systems. The latter are simulated by adding a given amount of a specific species to the system at a set time in the simulation. This allows us to simulate inputs coming from outside the system, such as injections made by the experimenter or signal coming from another DNA system present in the environment. Spikes, either unique or
periodic, can be added through the interface. Arbitrary inputs can also be loaded from an external file.

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 C) by an indirect one (A promotes B) 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 strand B, meaning that its concentration will also decrease slower (figure 3). 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 [21,33]. Those problems can be avoided by modifying parameters in the system (strand stability, template concentrations, enzyme activities, etc.), or making additional changes in the structure, operations that are easily carried out with DACCAD.

2. Methods

2.1. Mathematical model

To keep the model simple, yet descriptive enough, we model reactions at the domain level. Such level abstracts actual DNA sequences and instead considers only meaningful interactions, from the designer’s 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, because they have both an activation and an output site. This approach has proved itself over time in DSD systems [2,30,31], and can be extended to the DNA toolbox if we consider that enzyme activity is carried out as a single step. This means that no intermediate product of an enzymatic reaction, such as a partially extended DNA strand, can interact with other parts of the system. This also means that the evolution of the state of a module (activation or autocatalysis) over time can be derived only from the current concentrations of the relevant species and its current state (figure 4). With those restrictions, the set of possible reactions taking place is large, but straightforward: both signal and output strands can attach or detach to the template and an inhibition strand can displace them if there is a toehold [46] available, the polymerase enzyme will extend the signal strand, displacing the output if it was attached and so on. Possible reactions for a single template are shown in figure 5. This demonstrates one of the advantages of DACCAD: in a generic biochemical simulation program such as COPASI, 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, whereas it generates for COPASI five new species, 11 reactions and requires one 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 six new reactions have to be added to the mathematical model. Figure 6 shows the size difference between the representation of the same reaction network in DACCAD and COPASI.

The derivative of the concentration of a signal strand \( s \) is deduced from its interactions with all compatible modules as follows:

\[
\frac{ds}{dt} = \sum_{\text{temp}} \phi_{\text{in}}(t) + \sum_{\text{temp} \text{COPSI}} \phi_{\text{out}}(t) - \phi_{\text{exo}}(t) s(t),
\]

where \( I \) is the set of all modules accepting \( s \) as an activator, \( O \) is the set of all modules generating \( s \), \( \phi_{\text{in}}^{\text{temp}}(t) \) is the flow of \( s \) as input of a template temp, \( \phi_{\text{out}}^{\text{temp}}(t) \) is its contribution as the output of a template temp and \( \phi_{\text{exo}}(t) \) is 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. [3], we assume that exo mostly depends on the length of \( s \), which means it can take only 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 \( \text{exo} \) does not change over time. Also note that a given module can be both in \( I \) and \( O \) in the case of autocatalysis.

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

\[
\frac{df}{dt} = \phi_{\text{inh}}(t) + \sum_{\text{temp} \text{COPSI}} \phi_{\text{out}}(t) - \phi_{\text{exo}}(t) f(t).
\]

Note that there is only one flux for the inhibition term, because a given inhibition strand targets a specific module temp. \( \phi_{\text{in}}^{\text{temp}}(t) \), \( \phi_{\text{out}}^{\text{temp}}(t) \) and \( \phi_{\text{inh}}(t) \) can be derived directly from the current state of the module temp. Their expressions are given in the electronic supplementary material.

While the basic model does not take into account enzyme saturation, using first-order activity, such assumption is not realistic for a complex system. 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 term is further modified to take into account that multiple modules are competing for those resources [21]. Coaxial stacking, the collaborative stabilization that happens at a nick, and dangle are also considered by the model, as it affects the release speed of input and output. Explicit saturation expressions and a discussion on coaxial stacking are given in the electronic supplementary material.

It is possible to toggle the saturation for part or all of the enzymes (polymerase, nicking enzyme and exonuclease) to obtain results closer to reality at a (light) cost in computation time.

It is also possible to toggle some additional effects: 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 enzyme and act as competitive inhibitors); allowing 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 leak). Saturation is toggled on by default, both to be closer to the actual in vitro behaviour and to help design systems specifically using this phenomenon, such as in a winner-take-all system [19,32,33].

2.2. Implementation

2.2.1. Graphical interface and graph manipulation

The main graphical interface of DACCAD is separated into two areas (figure 7): 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 [47] 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 able to draw inhibition arrows. The plotting of time traces is done by the JFreeChart library [48]. It is also possible to obtain an animated version of the graph, displaying the evolution of species concentrations and enzyme saturation, using a second renderer supporting size and colour changes.

2.2.2. Equation generation and solving

In §2.1, 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 [49] 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 [4] where two excluding parts of a system get very close in concentration and requires careful integration to obtain the correct dynamics. Default parameters are given in the electronic supplementary material.

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.

2.2.3. Local optimization

While trial and error is a simple way to optimize in silico a system’s behaviour, such as an oscillator’s amplitude or frequency, it might be inconvenient owing 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 optimization algorithm [50], to perform parameter optimization. The structure is kept as defined by the user to make optimization possible in a reasonable amount of time. Structure optimization can also be attempted, but requires adapted algorithms [51, 52]. The user can define which parameters can be optimized, 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 minimize 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 sets the parameters to the best fit.

3. Results

Here, we present various systems that were simulated using DACCAD, going from 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.

3.1. Simple systems

As a proof of concept, we implemented Montagne et al.’s Oligator [3] and Padirac et al.’s bistable circuit [4]. The time series were compared with those of the original articles. 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 8. 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 autocatalytic signals. For the Oligator, the system obtains 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. [4]. This diagram is strongly
influenced by the coaxial stacking values of the various templates (see the electronic supplementary material).

3.2. Simple system optimization

The Oligator oscillates only on a restricted area of the complete parameter space [3]. 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 9. In this particular case, a very good match was found, but specific behaviours may be impossible. In the case of troubles with the evolution, it is recommended to reduce the number of evolved parameters 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 11), 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 sequester an intermediary species of the oscillator, making it actually twice slower. Good parameters were found by excluding this parameter from the optimization as well.

3.3. Combined systems

Using the two previous building blocks, we can build advanced systems that would require hundreds of lines if the ODEs were written by the user. By cascading two bistable systems, we can obtain a two-bit counter (figure 10). If instead we chose to combine an extended Oligator and a bistable circuit, we can obtain a frequency divider (figure 11). 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, such as in the two-bit counter, many factors have to be taken into account when cascading systems. The most important is the load effect [53]: when adding a template taking a given strand as input, the sequestering of this strand will change the behaviour of the system of which it is part. 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.

3.4. Saturation-based effects

Kim et al. [19] 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 [32]. In the DNA toolbox, enzymes are perfect examples of such shared resources [33]. Because the standard experimental parameters are set to avoid saturation (and thus competition) as much as possible, the activity of one enzyme has to be reduced by an order of magnitude.

Figure 5. 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 tempalone (template alone), tempin (signal strand attached), tempout (output strand attached), tempboth (both signal and output strands attached), tempext (template completely double-stranded) and tempinhib (inhibited template). (a) Working of an activation template without inhibition. (b) Reactions related to inhibition. Toe represents the fact that signal and output strands invade an inhibited template at a slower rate, because they have a very short toehold. (Online version in colour.)

Figure 6. Comparison between the representation of a simple DNA toolbox autocatalytic 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. (Online version in colour.)
Figure 7. Display panel. 2. Data panel. 3. Parameters area. Sets species and template-relative parameters. 4. Graph of the system. Selected nodes are displayed in blue. 5. File menu. 6. Species menu. 7. Options. Sets general and enzymatic parameters. 8. Plot. Simulates and displays the behaviour of the system. (Online version in colour.)

Figure 8. (a,b) The Oligator [3], simulated with first-order enzymatic activity (a) and with enzymatic saturation and competitive coupling (b). With the chosen parameters, the amplitude and frequency of oscillation are modified, but the behaviour is otherwise unchanged. (c,d) 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 (d), the system saturates the enzymes and loses its bistable properties. Performing some optimizations on the enzyme parameters gives us some insights into this phenomenon: it turns out that with Padirac et al.’s [4] parameters, the nicking enzyme saturates first, causing this loss of bistability. (Online version in colour.)
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 12. Saturation can also be harnessed to generate various behaviours, such as oscillations (figure 13) in a system that would be stable otherwise.
3.5. 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 a secret combination that the other player tries to guess. We chose to have two possible symbols and a combination of length four. The design was done incrementally. First, we created a guess validation module, combining a bistable (memory), an input system (signal amplification) and a reporter. The response was optimized in a saturated environment using CMA-ES. We then combined four such modules and optimized again to cover all input possibilities. The whole system is shown in figure 14. The design strategy is explained in the electronic supplementary material. A sample game is shown in figure 15.

4. Conclusion

In this study, 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 [36] or DNA-Melt [37,38] and the design rules presented in Baccouche et al. [29]. While the value of some parameters, such as the release slowdown owing to coaxial stacking, can be hard to predict, the actual value can be measured through some basic experiments and then updated in DACCAD, allowing the quick correction of a design. Such parameters can also be slightly modified from their actual experimental value to encompass more specific or sequence-dependent effects [54,55]. Finding a way to fit such experimental parameters directly from fluorescence data could be an important future application.

Note also that the very strength of DACCAD, i.e. its black box handling of kinetic complexities, brings along a limitation: it can only work with pure polymerase–nickase activator–inhibitor systems, and other reactions, such as hairpin formation from the predator–prey system [15], are forbidden. However, handling of 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. In that sense, DACCAD can be seen as a compiler graph to SBML, allowing the algorithmic generation of DNA toolbox systems (for example, systems trying to solve instances of the 3-SAT problem [56]). Moreover, this generation is facilitated by its simple graph description file format.

We then showed how the software can be used to design systems of increasing complexity. In particular, one can also observe different behaviours, such as saturation-based oscillation or the winner-takes-all effect. Local optimization can also be performed by using the state-of-the-art algorithm CMA-ES. Moreover, the mathematical model can be easily re-used with other design techniques, such as evolutionary algorithms, which may help optimizing systems following multiple possible objectives [57], or find new design patterns to create complex systems [51,52]. It would also be interesting to investigate two-dimensional reaction–diffusion implementations of the DNA toolbox [16,17], 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 fast off-the-shelf reaction–diffusion simulation software, such as ReaDY (https://code.google.com/p/reaction-diffusion/).

![Figure 14. Implementation of the Mastermind game. State bistable circuits are shown in blue, possible inputs in green, reporter species in yellow and lock species in red.](image)

![Figure 15. 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.](image)
Acknowledgements. We thank Prof. Nicolas Bredeche, Dr Anthony Genot, Dr Adrien Padiac and Alexandre Baccouche for useful discussions, the anonymous reviewers and all the DACCAD beta testers, who provided useful insights into the organization of the software and priceless bug reports.

Endnote

1Java 6 is required. Minimum specifications are available online on Oracle’s website at http://www.oracle.com/technetwork/java/javase/config-417990.html.

References


