Towards an Optimal DNA-Templated Molecular Assembler

Jakob L. Andersen [1], Christoph Flamm [2], Martin M. Hanczyc [3], Daniel Merkle [1]
[1] Department of Mathematics and Computer Science, University of Southern Denmark, Denmark
[2] Institute for Theoretical Chemistry, University of Vienna, Austria
[3] Centre for Integrative Biology CIBIO, University of Trento, Italy
daniel@imada.sdu.dk, jlandersen@imada.sdu.dk
mhanczyc@gmail.com, xtof@tbi.univie.ac.at

Abstract

In DNA-templated synthesis, reactants are attached to DNA strands and complementary DNA strands are used to control the reaction towards a goal compound. This very general, simple, and still efficient approach has proven to be successful for the design of complex one-pot synthesis for a large variety of compounds. For a given goal compound many different synthesis plans may exist, and all of them can potentially be implemented with many different DNA-templated programs. This raises the issue of how to automatically infer optimal low-level programs based on a high-level synthesis plan or a goal compound only. In this paper we will introduce a computational approach for DNA-templated synthesis based on graph rewriting approaches and the systematic exploration of chemical spaces. We will use them for verification of correctness of real-world synthesis plans as well as to illustrate the non-triviality of finding an optimal DNA assembler program.

Introduction

The "ideal synthesis", according to Wender (Wender and Miller, 1993), may be defined as a simple, safe, environmentally-acceptable, resource-efficient one-step operation, that quantitatively yields the desired complex target molecule from the readily available starting materials. Of course, such an idealized synthesis does not exist for the majority of the synthetically interesting target molecules, however multi-step one-pot reactions approximate Wender's ideal synthesis quite closely.

Therefore considerable effort has been put into the design, development and execution of such one-pot, multi-step chemical and biochemical synthesis strategies (Broadwater et al., 2005; Wang et al., 2007; Lundberg et al., 2008; B.Ramachary and Jain, 2011; Denard et al., 2013). This reaction architecture restricts all chemical transformations in the same one-pot space often with sequential addition of reactants or catalysts. Without having to perform multiple discrete purification steps as per normal in synthesis protocols, time is saved as well as resources. If designed properly the one-pot multi step protocol will produce the products of interest while minimizing the undesirable side products common in traditional synthesis.

A design strategy for one-pot synthesis is to covalently attach a short oligomer of DNA to each reactant (Winssinger and Gorska, 2013). In one design, an additional DNA strand is then added to the system as a sequence specific polymeric support that is complementary to both reactant strands, bringing both reactants physically close to one another, see the framed box in the top left of Figure 7. This structured chemical environment increases the effective concentration of the reactants and has been demonstrated to effectively enhance the rate of the reaction towards the desired specific product (e.g., McKee et al. (2010)). Several important and distinct chemical reactions can be supported by this design strategy including heteropolymerization, photoligation, click chemistry, condensation, cycloaddition, and many others (see Gorska and Winssinger (2013)).

Recently we described a computational generative chemistry approach based on graph grammar to explore previously intractable complex chemical spaces (Andersen et al., 2013a). As a result we can propose potentially interesting chemical pathways such as synthesis pathways or autocatalytic loops that may be present in the complex mixture. To further define the tools needed to navigate complex chemical spaces, here we present a design strategy based on both one-pot multi-step synthesis and reactions templated by nucleic acid complementarity. We explore how specific chemical pathways may be designed in sequential and parallel steps when reactants are tagged with short oligomeric DNA. We take a computer science perspective on the design of the entire system, by compiling a list of instructions that when implemented in wet chemistry will produce desired outcomes as reaction products.

In the next two sections we will introduce to DNA-templated synthesis and necessary definitions for its formalization. A framework based on the combination of graph rewriting approaches and a strategy framework for chemical space exploration will be presented. In the results section we use the framework in order to verify correctness of a DNA-templated synthesis as presented in (McKee et al., 2010) and we analyze two further syntheses with a focus on optimality. In the last section we conclude and give future directions.

DNA Templated Synthesis

State-of-the-Art

While most of the experimental details and questions for the DNA-templated organic synthesis have been worked out over the last couple of years (for a recent review see (Gorska and Winssinger, 2013)), computational questions are largely open. However, several computational design tasks in the realm of DNA computation and related areas have been approached in the last decade: (1) the primer selection problem (Pearson et al., 1996) which seeks a minimal set of short DNA sequences which specifically bind to a target DNA strand with minimal cross-reactivity, an $\mathcal{NP}$-hard problem, (2) thermodynamic design of multi-stable nucleic acid molecules (Höner zu Siederdissen et al., 2013), (3) sequence design for ensembles of interacting nucleic acid molecules (Zadeh et al., 2011), (4) pathway design for the self-assembly of nucleic acid nano-objects (Yin et al., 2008), (5) the design of DNA interaction networks with defined temporal behavior (Baccouchea et al., 2014), (6) a programming language for composable DNA circuits based on strand displacement as the main computational mechanism has been introduced (Phillips and Cardelli, 2009; Cardelli, 2010). However no theoretical efforts have been undertaken so far to clarify how a synthesis plan (i.e., the instructions how a target molecule is constructed from starting materials) can be "compiled" into DNA-templated assembler programs. This leads to the question: does an optimal synthesis plan translate also in an optimal DNA-templated assembler program? This touches the problem of how many DNA tags should be used to optimize a DNA-templated assembler program, and if such a program does or does not create side products. These issues will be investigated in the following sections.

Definitions

Here we introduce necessary definitions for DNA-templated synthesis, please refer to the framed box of Figure 7 for an illustration. A DNA *domain* is a sequence of nucleotides (Cardelli, 2010). DNA plus (resp. minus) strands with single domains are denoted with lowercase variables $a, b, \ldots$ (resp. over-lined lowercase variables $\overline{a}, \overline{b}, \ldots$). In Figure 7 plus (resp. minus) strands are illustrated as lines with half arrows pointing to the right (resp. left). DNA strands composed of several chemically linked domains are denoted as a sequence of the corresponding DNA domain variables, where the sequence of domains is always given from 5'-end to the 3'-end, e.g., ab denotes a plus strand of the sequence of the two DNA domains a and b (the blue-red strand in Figure 7), while $\overline{ba}$ denotes a sequence of two domains on the complementary minus strand. A sequence of domains and its complement form a fully stacked helix, e.g., ab and $\overline{ba}$ form a helix. Non-covalent sequences of domains are indicated by a $|$ sign between the domains (e.g., $\overline{b}|\overline{a}$). In this paper an *instruction strand* as well as a *release strand* is a sequence of two domains. Without loss of generality instruction (resp. release) strands are always plus (resp. minus) strands. Compounds, i.e., reactants and products, are denoted with uppercase variables $A, B, \ldots$. Compounds can be attached to the 3'-end or 5'-end of a domain. Without loss of generality only minus strands are modified (i.e., only minus strands are DNA adapters for compounds). A domain (or DNA adapter, or DNA tag) modified with a compound (or more specifically a reactant, resp. product) will be called *compound-DNA* (or more specifically *reactant-DNA*, resp. *product-DNA*). If a modification of a domain is at its 5'-end, we will use a superscript prefix notation (e.g., ${}^{A}\overline{a}$ denotes a DNA strand $\overline{a}$ which has been modified at its 5'-end with compound A). If a modification is at its 3'-end we will use a superscript postfix notation (e.g., $\overline{a}^{A}$ denotes a DNA strand $\overline{a}$ which has been modified at its 3' end with a compound A).

A complex of instruction strand and one or two compound-DNAs is formally denoted as a pair, e.g., (ba, $\overline{a}^{A}|^{B}\overline{b}$) refers to an instruction strand ba which has the two compound-DNAs $\overline{a}^{A}$ and ${}^{B}\overline{b}$ bound to the instruction strand. Note that in this example the compounds A and B are in close vicinity. We conveniently use the special symbol ϵ for the empty strand or an empty molecule. I.e., the complex (ba, $\overline{a}^{A}|\epsilon$) has an unbound domain b and the complex (ba, $\overline{a}^{A}|^{\epsilon}\overline{b}$) has a domain $\overline{b}$ attached to b and $\overline{b}$ has no compound attached. Note, that ${}^{\epsilon}\overline{b}$ refers to the same compound as $\overline{b}^{\epsilon}$.

An example of how we depict reactions and compound-DNAs is given in the framed area of Figure 7: given the two reactant-DNAs ($\overline{b}^{B}$ and ${}^{A}\overline{a}$) and the instruction strand ba, a complex (ba, $\overline{a}^{A}|^{B}\overline{b}$) is formed via a reaction (i.e., a hyperedge) which is illustrated as rectangle.

Graph Grammars, Generative Chemistries, and Strategies

While several methods for reasoning about chemistry (Dittrich et al., 2001) have been studied, we promote graph grammar approaches (Benkö et al., 2003; Rozenberg and Ehrig, 1997) as the core formal framework to handle chemical transformations. Graph grammars naturally capture the algebraic nature of chemical reactions where molecular graphs operate upon each other to produce (potentially) novel molecular graphs. Molecules are abstracted to edge and vertex labeled graphs while reactions are expressed as graph rewrite rules between input and output molecular graphs. The so-called Double Pushout (DPO) (Rozenberg and Ehrig, 1997) approach provides the most intuitive direct encoding of chemical reactions and the closest connection to the language of chemistry. A DPO transformation rule $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ consists of three graphs L, R and K known as the left, right and context graph, respectively, and two graph morphisms l and r that determine how the context is embedded in the left and the right graph. The rule p can be applied to a graph G if the left graph L can be found in

G and some additional consistency conditions are satisfied.

Graph grammars can be thought of as context-sensitive grammars, with strings replaced by labeled graphs. Repeated application of a set of reaction rules to a set of chemical units generates the "language" of all possible chemical units reachable from the initial setup. Chemical units can refer to molecules, DNA strands, complexes of instruction strand and compound-DNA, or a combination of such structures. The modeling of DNA strands and compound DNAs as undirected node- and edge-labeled graphs and the modeling of operations on these chemical units via DPO is straightforward and will be shown based on an example. In Figure 1(a) a graph grammar production rule $p\colon (L \to R)$ is illustrated (for simplicity we omit the context graph K). The diagram shows the labeled subgraphs L (two components) and R (one component), the host graphs G and H, the subgraph matching morphisms (downward arrows), and the production morphism (right arrows) for a chemical operation. In this case a graph grammar rule in order to bind compound-DNA $\overline{a}^A$ to an instruction strand ba is shown, denoted as $G \overset{p}{\Rightarrow} H$. Note, that the production could be applied to any other instruction strand, too, as long as L is found as subgraph in the host graph G. Figure 1(b) shows the semantically identical but prettified version, which will be used throughout the paper.

The chemical units and their producing reactions are most conveniently organized in a hypergraph, i.e., the chemical space. While graph theoretical methods are well established for prediction of chemo-physical properties of individual molecules (Gramatica, 2007; Le et al., 2012), analysis of the underlying hypergraph, i.e., the chemical space is, with a few exceptions, lacking. In (Andersen et al., 2012), the NP-completeness of maximizing the production of a desired compound in a given chemical reaction network was proven. The mathematical definition of functional sub-networks, such as a synthesis pathway as flow problems on hypergraphs, allows to detect relevant chemical transformation motifs, e.g., potential synthesis plans in a chemical space.

Recently a generic approach for composing graph grammar rules to define chemically useful rule compositions was introduced (Andersen et al., 2013b). The idea of rule composition has been utilized to define an efficient framework for defining strategies to systematically explore chemical spaces (Andersen et al., 2014). An analysis of the complex chemistry of hydrogen cyanide was recently published in (Andersen et al., 2013a), which uses this framework. Here, we will define high-level constructs for DNA strand computing based on this strategy framework, and we therefore briefly describe a simplified version of it.

Exploration of a chemical space proceeds step-wise. The state of an exploration is a set of molecular graphs (including product-DNAs, complexes, and instruction strands in the case of DNA-templated computing). *Exploration strategies*

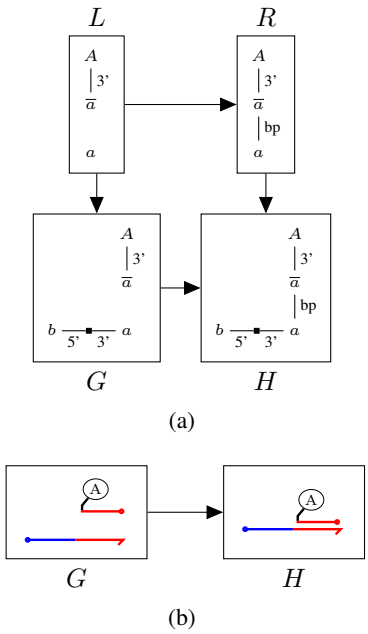

Figure 1: Graph Grammar rule that illustrates a production rule p for binding of a compound-DNA $\overline{a}^A$ to an instruction strand that is composed of the plus strand a and other strands. Applying the production rule p to graph G produces H, denoted as $G \overset{p}{\Rightarrow} H$.

are functions from and to such states. A strategy is defined by a set P of transformation rules in the DPO formalism. The *core strategy* will apply all the rules to all combinations of graphs in the input state. The resulting state consist of all new graph derived and all graph which have not been an educt in any rule application. Formally, for an input state F the resulting state F' is given by

$$F' = \{h \in H \mid \forall G \subseteq F, p \in P : G \overset{p}{\Rightarrow} H\} \cup \{g \in F \mid \forall G \subseteq F, g \in G, p \in P : G \overset{p}{\not\Rightarrow} H\}$$

As shorthand we write this as $F' = P(F)$.[1] To support the dynamic addition of chemical units as needed in DNA-templated synthesis, an *addition strategy* is used, which, given a state F, adds a molecule graph g and yields the state F', i.e., $F' = F \cup \{g\}$. We write this as $F' = \mathbf{add}[g](F)$. The last strategy we will use for DNA strand computation is the *composition strategy*, which corresponds to function composition; given strategies Q_1 and Q_2 their composition $Q_2 \circ Q_1$ is a new strategy. To increase left-to-right readability for large composition chains we opt to write it as $Q_1 \to Q_2$. Instructions of a DNA program as defined in the next section will be based on this formalism.

[1] In relation to (Andersen et al., 2014), this is an abstraction of the strategy $\mathbf{revive}[\mathbf{parallel}[P]]$.

Computational Framework for DNA Templated Synthesis

In this section we will combine the graph grammar approaches for DNA-templated synthesis and the strategy framework in order to define a computational framework for artificial DNA-templated synthesis. The overall structure of the framework is as follows. Based on graph rewriting approaches a chemical space is iteratively expanded (without any DNA tagging), formally this leads to a hypergraph (also called derivation graph DG). Note, that for the artificial DNA-templated synthesis this derivation graph might also be given beforehand. The nodes in DG correspond to chemical compounds and the directed hyperedges correspond to chemical reactions. Within the chemical space DG we infer a pathway that corresponds to a synthesis plan that we are aiming to realize based on DNA-templated synthesis.

To the best of our knowledge, graph rewriting approaches for DNA-templated synthesis do not exist. We use the same graph rewriting framework to augment the chemical compounds of a chemical space with DNA adapters and iteratively infer compounds (now including DNA tagging, cmp. previous section) controlled by a given DNA-templated (synthesis) assembler program (or just DNA program). Besides the derivation graph DG we implicitly assume that the mapping of the tags for a DNA-templated reaction is given, i.e., for each tagged version of a reaction $A + B \rightarrow C + D$ in DG the information is known, if the DNA strand attached to A (resp. B) will be attached to C or D on the product side.

An advantage of using the same framework for expanding the chemical space of compound-DNAs, DNA plus and minus strands, instruction strands, and the corresponding complexes is that this allows for a straightforward and easy integration of our existing graph rewriting approaches. Consequently this means that, e.g., an atom-to-atom mapping is known for all reactions involved. Furthermore, identical methods for pathway inference on the different or integrated chemical spaces could be used. In order to "execute" a given DNA-templated assembler program we will make heavy use of a simplified version of our strategy framework as presented in (Andersen et al., 2014) and outlines in the previous section. If the products after termination of the DNA program do not correspond to the products wished, then we have an unwanted side-effect, i.e., side products we were not aiming for.

Programs for DNA-templated synthesis

DNA programs will be defined in Python[2], which is used as an interface language to a C++ library in which all the presented approaches are implemented. In our model a program is an exploration strategy, and an empty program is the identity strategy. Each operation in the program adds a specific strategy by composition. Let the current program be the strategy Q, then the following describes the valid operations.

- `monomer(compound, DNA-tag, modific. end)`:
  A compound-DNA corresponding to the arguments of the operation is added to the exploration state, e.g.,

  monomer("A", **tag**="$\overline{a}$", **end**="5")

  Formally this corresponds to the strategy composition $Q := Q \rightarrow \mathbf{add}[^{A}\overline{a}]$.
- `instruction(strand1, strand2)`:
  Semantically this means, that an instruction strand is added to the current state, e.g.,

  instruction("b", "a")

  Formally this means $Q := Q \rightarrow \mathbf{add}[ba]$.
- `release(strand1, strand2)`:
  A release strand is added to the current state, e.g.,

  release("$\overline{a}$", "$\overline{b}$")

  Formally this means $Q := Q \rightarrow \mathbf{add}[\overline{ab}]$

These operations are grouped into sets of *add* operators, e.g.,

```
add(monomer("A", tag="ā", end="5"),
    instruction("b", "a"))
```

where each *add* operation implicitly ends with a reaction strategy Q_{react}. This strategy implements the details of rule application for strand displacement, strand binding, and reactions between monomers from the original chemistry. E.g., the *add* operation above means $Q := Q \rightarrow \mathbf{add}[^{A}\overline{a}] \rightarrow \mathbf{add}[ba] \rightarrow Q_{\text{react}}$. Running a DNA-program corresponds to evaluating the strategy Q on the empty state. The resulting state will be used for *DNA program verification*, i.e., if unwanted products are in the final state, then the program is not *side-effect free*.

Results

We present three different setups for DNA program verification. The first example is based on an artificial chemical space and an artificial synthesis plan with four reactants. The second example is a real-world example that illustrates the correctness of the wet-lab results presented in (McKee et al., 2010). The third non-trivial example illustrates how even simple chemical spaces might lead to complicated DNA programs if the program should fulfill a certain optimality criterion.

Synthesis 1: Four Reactants and Four Adapters

In this example we aim at synthesizing product X based on four given reactants A, B, C, and D. A chemical space as depicted in Figure 2 is assumed. In this case the only possible synthesis plan to be implemented with DNA-templated synthesis for compound X is based on the reactions $A+B \rightarrow E, C+D \rightarrow F, E+F \rightarrow X$. This synthesis plan is highlighted with bold lines in the chemical space in

[2] overlined strings will be used for convenience

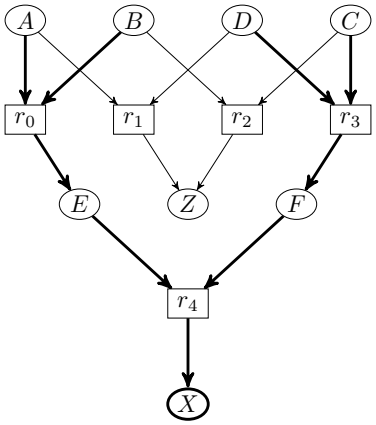

Figure 2: Artificial Chemical Space / Derivation Graph; the synthesis plan to be implemented based on DNA-templated synthesis is highlighted (bold).

```
Input: A Derivation Graph DG
Output: A Product Set

comp = DNAComputer(dg)
prog = comp.makeProgram()
prog.add(monomer("A", tag="ā", end="3"),
         monomer("B", tag="b̄", end="5"),
         monomer("C", tag="c̄", end="3"),
         monomer("D", tag="d̄", end="5"),
         instruction("b", "a"),
         instruction("d", "c"))
prog.add(release("ā", "b̄"),
         release("c̄", "d̄")
prog.add(instruction("b", "c"))
prog.add(release("c̄", "b̄"))
prog.run()
return prog.products()
```

Figure 3: DNA-templated program for the synthesis plan (four reactants) given in Fig. 2); four different DNA tags attached to the educts are used.

Figure 2. We assume that for each reaction in Figure 2 the mapping of the strands from educts to products is given, i.e., (as in our case there is only one product for each reaction), the information is given if for a reaction $A + B \rightarrow C$, the product-DNA of C will be attached to the DNA strand that was attached to A or C. Without loss of generality we assume that the product-DNA of C will inherit the DNA strand attached to A, for reaction $C + D \rightarrow E$ the product-DNA of E will inherit the DNA strand attached to C and for reaction $E + F \rightarrow X$ the product-DNA of X will inherit the DNA strand attached to F. Note, that the chemical space augmented with DNA adapters is much larger; allowing 2 different adapters only leads to a space of 2928 nodes and 8324 reactions even for this toy example, due to the combinatorial explosion of how to build compound-DNAs and complexes of them.

Using four DNA adapters a side-product free one-pot synthesis is straightforward. The monomers A (resp. C) are attached to the 3'-end of the DNA strand $\overline{a}$ (resp. $\overline{c}$) creating the reactant-DNA $\overline{a}^A$ (resp. $\overline{c}^C$). The monomers B (resp. D) are attached to the 5'-end of the DNA strands b (resp. d) creating complexes ${}^B\overline{b}$ (resp. ${}^D\overline{d}$). Using the instruction strand ba, the reactant-DNA $\overline{a}^A$ and ${}^B\overline{b}$ will form the complex $(ba, \overline{a}^A|{}^B\overline{b})$. Reactants A and B are now in close proximity and will react, and the complex $(ba, \overline{a}^\epsilon|{}^E\overline{b})$ will be formed. At the same when ba is added, the instruction strand dc is added, which in parallel and without interference will form the complex $(dc, \overline{c}^F|^\epsilon\overline{d})$. The next step will release the product-DNAs from their instruction strand, which is done by adding DNA-strands $\overline{ab}$ and $\overline{cd}$. This will release ${}^E\overline{b}$ and $\overline{c}^F$ and also form the non-reactive complexes $(ba, \overline{ab})$ and $(dc, \overline{cd})$. Subsequently, a newly added instruction strand dc will form the complex $(dc, \overline{c}^F|{}^E\overline{d})$, and E and F will react to X, forming the complex $(dc, \overline{c}^X|^\epsilon\overline{d})$. After releasing the reactant-DNA finally the goal product-DNA $\overline{c}^X$ is available and could be easily extracted from this one-pot synthesis plan via the unique tag. Note, that no side products were formed (i.e., all atoms from the educts ended up in the single goal compound X). The overall 4 step DNA program, is shown in Figure 3.

Synthesis 2: Sequence-Controlled Oligomers

In this section we analyze a real-world one-pot multi-step synthesis as presented in (McKee et al., 2010). Using two different mechanisms for the synthesis of oligomers they presented how i.) both mechanisms individually can be used to synthesize specific 4-mers, and ii.) how a specific 6-mer is created by coupling two 3-mers, which have been synthesized in parallel by both mechanisms. In the alternating-strand (resp. same-strand) mechanism the growing strand of the synthesis is transferred (resp. not transferred) between DNA adapters in single steps. More specifically, i.) the alternating-strand mechanism is initiated with a mono-functional ylide monomer (FAM), for which at each step the growing chain is transferred to the incoming monomer. This reaction is stopped by a transfer to a mono-functional aldehyde monomer (BAL); ii.) the same-strand mechanism is initiated with an aldehyde monomer and the growing chain remains on the same DNA adapter throughout the synthesis. The reaction in this case is terminated by addition of an ylide monomer (FAM). For details on determining the tagging of the monomers (all monomers are uniquely tagged), for determining the instruction and release strands, and for the sequence of the addition of the strands we follow precisely the description of (McKee et al., 2010), which we would classify as a straightforward synthesis similar to the artificial example as presented in the first example of the results section. From (McKee et al., 2010) we verified the side-effect free synthesis of the 6-mer as well as the side-effect free synthesis of the 4-mers, due to space limitations we will focus on the synthesis of a 4-mer.

Chemical Space: The derivation graph based on all possible 2-to-1 reactions of the four monomers has 27 nodes and 47 reactions. The goal compound for the synthesis plan is the

```
1  Input: A Derivation Graph DG
2  Output: A Product Set
3
4  prog = comp.makeProgram()
5  prog.add(monomer("A", tag="ā", end="3"),
6           monomer("B", tag="b̄", end="5"),
7           instruction("b", "a"))
8  prog.add(monomer("C", tag="ā", end="3"),
9           monomer("D", tag="b̄", end="5"),
10          instruction("b", "a"))
11 prog.add(release("ā","b̄"))
12 prog.add(instruction("b", "a"))
13 prog.add(release("ā","b̄"))
14
15 prog.run()
16 return prog.products()
```

Figure 4: DNA program for the synthesis plan given in Figure 2); only two different DNA tags attached to the educts are used; the program has unwanted side-effects.

```
1  Input: A Derivation Graph DG
2  Output: A Product Set
3
4  prog = comp.makeProgram()
5  prog.add(monomer("A", tag="ā", end="H"),
6           monomer("B", tag="b̄", end="T"),
7           instruction("b", "a"))
8  prog.add(release("b", "a"))
9  prog.add(instruction("b", "c"),
10          instruction("c", "a"))
11 prog.add(monomer("C", tag="ā", end="H"),
12          monomer("D", tag="b̄", end="T"),
13          instruction("b", "a"))
14 prog.add(release("b", "a"))
15 prog.add(instruction("c", "b"))
16 prog.add(release("b", "c"))
17 prog.add(instruction("b", "a"))
18 prog.add(release("b", "a"))
19
20 prog.run()
21 return prog.products()
```

Figure 5: DNA program for the synthesis plan given in Figure 2); again, two different DNA tags attached to the educts are used; the program has *no* unwanted side-effects.

the oligomer FAM-ALA-PHE-BAL (denoted as F-A-P-B). There are 5 synthesis plans to produce the goal compound, of which the four of them are linear. For the mapping of the tags of educt-DNA to the product-DNA for DNA-templated reactions we straightforwardly follow the descriptions from (McKee et al., 2010).

DNA Program and verification: We omit the source code of the DNA program due to space limitations. However, the program can easily be inferred based on Figure 6, which depicts the automatically inferred sequence of states of the program. The linear synthesis plan is implemented with a 6-step DNA program for templated synthesis. After termination of the program there exist several products. However, the only product-DNA created is the goal compound, therefore the linear synthesis plan is indeed side-effect free (in Figure 6 all products are highlighted with bold lines).

Synthesis 3: Four Reactants and Two Adapters

Both previous examples used unique DNA adapters for all the educts, i.e., the number of educts was identical to the number of different tags attached to the educts. In order to reduce the number of tags (and therefore increase their diversity) to an optimal amount, the DNA programs need a significant rewrite. In the following example the goal is to find a DNA program for the synthesis plan from Figure 2 with less than four different tags as used in the first example of the results section. It is not hard to see, that it is not possible to use one tag only. For the chemical space and synthesis plan to be implemented, please refer to Figure 2.

Naïve Approach: A DNA program using two tags only is shown in Figure 4. It follows the same idea as the program from Figure 3. However, when the four reactant-DNA $\overline{a}^A$, ${}^B\overline{b}$, $\overline{a}^C$, and ${}^D\overline{b}$ are added, the instruction strand ba would lead to side-effect as, e.g., $\overline{a}^A$ and ${}^D\overline{b}$ would form the complex $(ba, \overline{a}^A|{}^D\overline{b})$, and A and D would react to the unwanted compound Z. While this can be circumvented by sequentially adding instruction strands, the program will still not be side-effect free. After release (line 11) there will be the compound-DNAs ${}^E\overline{b}$ and $\overline{a}^F$ in the pot, but also minus strands $\overline{a}$ and $\overline{b}$. These will bind to the instruction strand added (line 12) and form, e.g., the complex $(ba, \overline{a}^\epsilon|{}^E\overline{b})$. The release operation (line 13) will release the goal compound, but in addition also the unwanted side products ${}^E\overline{b}$ and $\overline{a}^F$.

Correct Approach: A correct program is shown in Figure 5. An illustration of the correctness of the 9-step program is given in the Figure 7. After execution, the only compound-DNA with a non-empty compound attached is the goal compound (step 9 in Figure 7). Similar to classical synthesis this goal is reached by disabling DNA strands that would otherwise indirectly lead to unwanted side products as in the naïve approach. In step 2 of Figure 7, the empty DNA adapter ($\overline{a}$ in the program, depicted red in the figure) that was originally attached to A is disabled by attaching it to an instruction strand (ca in the program, depicted orange-red in the figure). Similar in step 5 the adapter originally attached to D ($\overline{b}$ in the program, depicted in blue) is disabled by an instruction strand (cb in the program, orange-blue in the figure). Another important mechanism used is the temporal protection of compound-DNA ${}^E\overline{b}$ by adding the instruction strand bc (step 2 in Figure 7). This compound-DNA is released again later (step 6 in Figure 7, line 16 in the program) in order to further react.

Conclusion and Future Directions

We introduced an approach for DNA-templated synthesis based on graph grammar approaches. We illustrated important steps towards an organic synthesis compiler which translates the high-level description of an synthesis plan into an executable DNA-templated assembler program. In particular we showed that even the verification of the correctness of a given DNA assembler program is already a non trivial problem. In particular the number of DNA tags has a huge

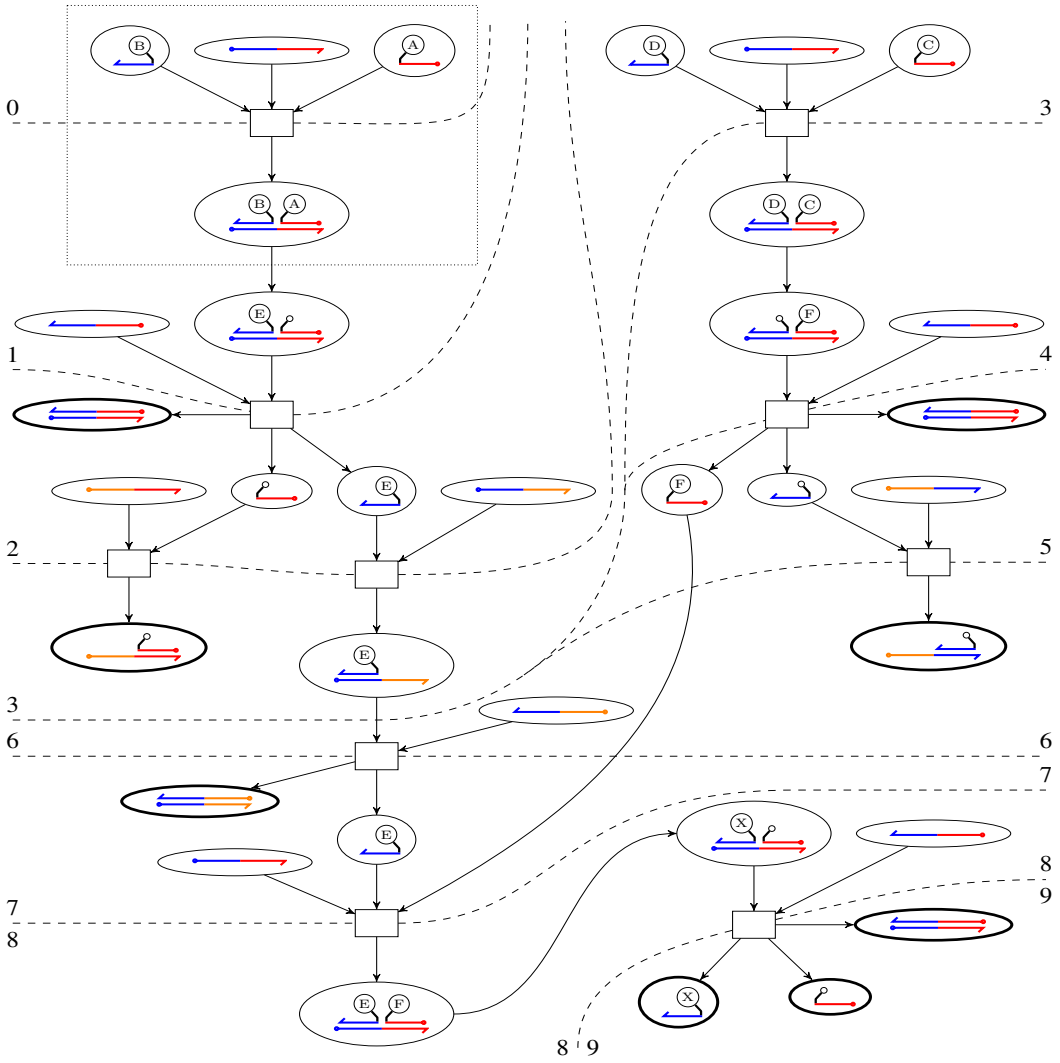

Figure 7: 9-step DNA-templated one-pot synthesis for compound X from Figure 2 using the DNA program from Figure 5; all the final products are indicated with bold lines, no unwanted side-effect exist; dashed lines partition the molecules into the individual states of the mixture between computation/synthesis steps.

impact on the length and complexity of the resulting DNA-templated assembler program. Here an expansion of the instruction set via strands which can form hairpin structures depending on the context with reactants hooked up to both ends could probably lessen the number of needed DNA tags. Furthermore, automatic inference of (optimal) DNA assembly program for the synthesis of a specific compound is a major line of our further research. A central question for the future is how optimality in the synthesis plan translates to optimality in the DNA assembler program. It is likely that optimality in these two formulations are competing qualities and therefore the problem must be attacked by a multi-objective optimization approach. Here the fact that our approach is completely formulated in the language of graph grammars is a clear advantage.

Acknowledgements

This work was supported in part by the EU-FET grants RiboNets 323987 and EVOBLISS 611640,, the COST Action CM1304 “Emergence and Evolution of Complex Chemical Systems”, and the Danish Council for Independent Research, Natural Sciences.

References

Andersen, J. L., Andersen, T., Flamm, C., Hanczyc, M. M., Merkle, D., and Stadler, P. F. (2013a). Navigating the chemical space of HCN polymerization and hydrolysis: Guiding graph grammars by mass spectrometry data. *Entropy*, 15(10):4066–4083.

Andersen, J. L., Flamm, C., Merkle, D., and Stadler, P. F. (2012). Maximizing Output and Recognizing Autocatalysis in Chemical Reaction Networks is NP-Complete. *Journal of Systems Chemistry*, 3(1).

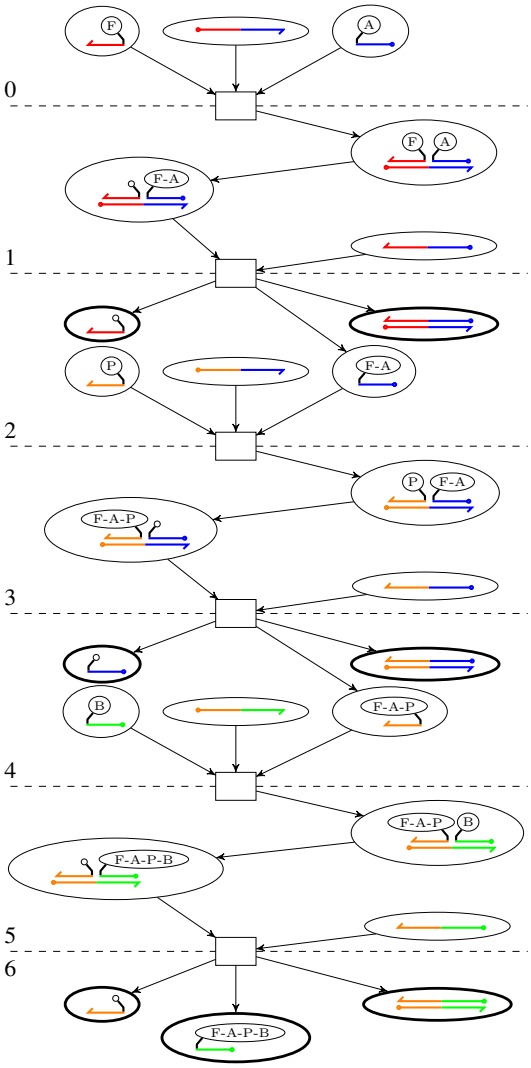

Figure 6: 6-step DNA-templated one-pot synthesis for the oligomer F-A-P-B (cmp. McKee et al. (2010)); all the final products are indicated with bold lines, no unwanted side-effect exist.

Andersen, J. L., Flamm, C., Merkle, D., and Stadler, P. F. (2013b). Inferring chemical reaction patterns using rule composition in graph grammars. *Journal of Systems Chemistry*, 4(1):4.

Andersen, J. L., Flamm, C., Merkle, D., and Stadler, P. F. (2014). Generic strategies for chemical space exploration. *International Journal of Computational Biology and Drug Design*. in press, TR: `http://arxiv.org/abs/1302.4006`.

Baccouchea, A., Montagnec, K., Padiraca, A., Fujiia, T., and Rondeleza, Y. (2014). Dynamic DNA-toolbox reaction circuits: A walkthrough. *Methods*. doi: 10.1016/j.ymeth.2014.01.015.

Benkö, G., Flamm, C., and Stadler, P. F. (2003). A graph-based toy model of chemistry. *Journal of Chemical Information and Computer Science*, 43(4):1085 – 1093.

B.Ramachary, D. and Jain, S. (2011). Sequential one-pot combination of multi-component and multi-catalysis cascade reactions: an emerging technology in organic synthesis. *Organic & biomolecular chemistry*, 9(5):1277–1300.

Broadwater, S. J., Roth, S. L., Price, K. E., Kobaslija, M., and McQuade, D. T. (2005). One-pot multi-step synthesis: a challenge spawning innovation. *Org. Biomol. Chem.*, 3(16):2899–2906.

Cardelli, L. (2010). Two-domain dna strand displacement. In *DCM*, pages 47–61.

Denard, C. A., Hartwig, J. F., , and Zhao, H. (2013). Multistep one-pot reactions combining biocatalysts and chemical catalysts for asymmetric synthesis. *ACS Catalysis*, 12(3):2856–2864.

Dittrich, P., Ziegler, J., and Banzhaf, W. (2001). Artificial chemistries — a review. *Artifical Life*, 7:225 – 275.

Gorska, K. and Winssinger, N. (2013). Reactions templated by nucleic acids: More ways to translate oligonucleotidebased instructions into emerging function. *Angewandte Chemie International Edition*, 52(27):6820–6843.

Gramatica, P. (2007). Principles of QSAR models validation: internal and external. *QSAR & Combinatorial Science*, 26(5):694–770.

Höner zu Siederdissen, C., Hammer, S., Abfalter, I., Hofacker, I. L., Flamm, C., and Stadler, P. F. (2013). Computational design of RNAs with complex energy landscapes. *Biopolymeres*, 99(12):1124–1136.

Le, T., Epa, V. C., Burden, F. R., and Winkler, D. A. (2012). Quantitative structure-property relationship modeling of diverse materials properties. *Chem. Rev.*, 112:2889–2919.

Lundberg, P., Hawker, C. J., Hult, A., and Malkoch, M. (2008). Click assisted one-pot multi-step reactions in polymer science : Accelerated synthetic protocols. *Macromolecular rapid communications*, 29(12–13):998–1015.

McKee, M., Milnes, P., Stulz, E., Turberfield, A., and O'Reilly, R. (2010). Multi-step DNA templated reactions for the synthesis of functional sequence controlled oligomers. *Angew. Chem. Int. Ed.*, 49:7948–7951.

Pearson, W. R., Robins, G., Wrege, D. E., and Zhang, T. (1996). On the primer selection problem in polymerase chain reaction experiments. *Discr. Appl. Math.*, 71(1-3):231–246.

Phillips, A. and Cardelli, L. (2009). A programming language for composable DNA circuits. *J. R. Soc.*, 6(Suppl 4):S419–S436.

Rozenberg, G. and Ehrig, H. (1997). *Handbook of graph grammars and computing by graph transformation*, volume 1. World Scientific.

Wang, Y., Ye, X.-S., and Zhang, L.-H. (2007). Oligosaccharide assembly by one-pot multi-step strategy. *Org. Biomol. Chem.*, 5(14):2189–2200.

Wender, P. A. and Miller, B. L. (1993). *Towards the Ideal Synthesis: Connectivity analysis and multibond-forming processes*, volume 2 of *Organic Synthesis: Theory and Applications*, pages 27–66. JAI Press, Greenwich CT.

Yin, P., Choi, H. M. T., Calvert, C. R., and Pierce, N. A. (2008). Programming biomolecular self-assembly pathways. *Nature*, 451:318–322.

Zadeh, J. N., Steenberg, C. D., Bois, J. S., Wolfe, B. R., Pierce, M. B., Khan, A. R., Dirks, R. M., and Pierce, N. A. (2011). NUPACK: Analysis and design of nucleic acid systems. *J. Comput. Chem.*, 32(1):170–173.