DNA-Templated Synthesis Optimization

Bjarke N. Hansen, Kim S. Larsen, Daniel Merkle, and Alexei Mihalchuk

Dept. of Mathematics and Computer Science, University of Southern Denmark,
Odense, Denmark. {kslarsen,daniel}@imada.sdu.dk

Abstract. In chemistry, synthesis is the process in which a target compound is produced in a step-wise manner from given base compounds. A recent, promising approach for carrying out these reactions is DNA-templated synthesis, since, as opposed to more traditional methods, this approach leads to a much higher effective molarity and makes much desired one-pot synthesis possible. With this method, compounds are tagged with DNA sequences and reactions can be controlled by bringing two compounds together via their tags. This leads to new cost optimization problems of minimizing the number of different tags or strands to be used under various conditions. We identify relevant optimization criteria, provide the first computational approach to automatically inferring DNA-templated programs, and obtain optimal and near-optimal results.

1 Introduction

The first instance where DNA has been used to execute an algorithm in order to solve a combinatorial optimization problem dates back to 1994. In [1], Adleman demonstrated how a small instance of the Hamiltonian Path Problem could be solved using DNA sequences. Since then, DNA nanotechnology has been used as a powerful tool for a wide variety of research and engineering questions. Examples include polyhedral mesh rendering, where DNA sequences are designed such that they fold into predefined complex three-dimensional structures [3], and design of DNA-based molecular motors that can be used to transport cargo molecules [16]. Appealing features of DNA-based designs is their programmability, the inherent concurrency, the predictability, and the fact that DNA sequences are relatively cheap and easy to synthesize. The number of approaches utilizing DNA-based chemistry as a source for the discovery and the design of novel drug-like molecules has increased rapidly in recent years [7]. Basically all large pharmaceutical companies have already started utilizing this technology. DNA-based chemistry approaches include a method called DNA-templated organic synthesis [12], where the goal is to synthesize an organic compound in a step-wise manner. In an individual step of a synthesis plan [11], either two compounds are combined (affixation reaction) or a single compound is modified (cyclization reaction). This information can be captured in a rooted unary-binary tree, though often cyclization reactions can be ignored from a combinatorial point of view, making the tree binary. Chemists are aiming at *efficient* synthesis (the yield of all reactions and therefore the yield of the overall process should be high) and *one-pot* synthesis (for instance, avoiding complicated separation and purification processes based on contaminating compounds that require subsequent extraction of a specific product from a mixture of compounds).

In DNA-templated synthesis, the base compounds are "tagged" with DNA sequences. These tags are used to bring the compounds in close vicinity (and thereby react). This is done by adding a complementary DNA strand, called an instruction strand, which is a concatenation of the complementary strands of the two tags that are attached to the base compounds. In contrast to classical synthesis approaches, DNA-templated synthesis allows for much lower concentrations of reactants due to the tagging, which leads to a dramatically increased effective molarity. We refer to [12] and [8] for in-depth reviews and specifically [13] and [10] for examples of successful, non-trivial, multi-step DNA-templated molecule syntheses.

The synthesis tree together with a specification of how to tag the base compounds and according to which topological ordering of the tree the reactions should be carried out defines a so-called DNA-templated program. While high-level formalisms for DNA computational structures have been studied [14,4] before, there are no prior attempts to automatically inferring DNA-templated programs based on a given synthesis tree. In [2], graph rewriting approaches have been used for verifying correctness of given DNA-templated programs, but neither were programs automatically inferred nor optimization questions answered. With careful choice of tagging and topological ordering, it is possible to use the same tags and strands repeatedly, which leads to the optimization problems we consider. To avoid unintended interference, tags and strands that should be different must be some mimimum edit distance away from each other. If one uses too many different tags or strands, these must be made longer in order to obtain this, leading to higher production costs.

Another cost stems from the tagging of chemical compounds, which is a somewhat sophisticated chemical procedure. Thus, while it is interesting to minimize the use of different tags and strands in general, it is also interesting just to minimize the number of different tags used on the base compounds.

We present i) optimal or near-optimal methods for minimizing the number of strands, ii) a somewhat more involved method for minimizing the number of strands and subsidiarily the number of tags, iii) a method for minimizing the number of strands when only two different tags are allowed on base compounds, but longer programs using blocking are allowed, and, finally, iv) a generic ILP formulation of the optimization problems which is then without time complexity bounds.

2 Modeling DNA-Templated Synthesis

The goal of this section is to present a model for DNA-templated synthesis such that we can work with these issues in a combinatorial manner. We identify some basic operations and restrictions on how these can be applied, with the goals in mind. We would like to emphasize that we do not make any simplifying assumptions, preventing our solutions from leading to programs that can be realized chemically. However, there could be other choices of computational units and goals, and our focus is on presenting an initial model that is as simple as possible while still capturing the fundamental chemical intricacies. Our description will lead to a definition of the input, available operations, constraints, and a number of optimization objectives.

From a chemist, we get *a synthesis tree*, which we assume is binary, where the *leaves represent compounds.* We refer to these as the *base* compounds. The tree can be interpreted as a recipe in the following manner. Each leaf of the tree represents an existing base compound. Now, we bring compounds to react in an order respecting the tree structure. Thus, first the compounds corresponding to two leaves are made to react, resulting in a new compound, which we refer to as an *intermediate* compound. We keep going until we reach the root, and have at that point produced our final *target* compound. The order of combining the compounds should simply be a topological ordering of the tree. We draw the trees with the root at the bottom, as it is usually done for synthesis trees, and hope our fellow computer scientists can accept this normality.

We detail the operations below. Our textual description is complete and self-contained, but it might be helpful to refer to the appendix for an example program. In order for two compounds to react, they must be in close proximity, and two compounds do not react if they are distant enough. To obtain proximity, the compounds are equipped (tagged) with DNA sequences, and the compound is at one of the two ends of the sequence. We refer to such a DNA sequence on a compound as a *tag* and choose an orientation so that we can refer to the left and right ends of a tag. Assume x and y are the tags of compounds X and Y, respectively, and X is at the right end of x and Y at the left end of y. If we add the *complementary* strand of the concatenation of x and y, denoted $\overline{xy}$, x and y will attach to the $\overline{x}$-part and $\overline{y}$-part of $\overline{xy}$, respectively, bringing X and Y close together and the reaction of X and Y takes place. We refer to such a strand as an *instruction* strand and the process as a *react* operation. In the above, and in the rest of the paper, when we refer to a strand, it can always be thought of as the concatenation of two tags. The resulting intermediate compound will lose one of its tags in the process of the reaction and will thus afterwards be tagged with either x or y in a deterministic fashion decided by the compounds, i.e., along with the synthesis tree, a chemist will tell us, for every internal node, which of the two tags from the child nodes will be the tag of the produced intermediate compound. We say that the node *inherits* the tag from the child in question and we may use a bold edge to indicate this. This annotated tree forms our input from the chemist. Note that the compounds and what they become when they react is not important to us; only the tags (and how they are attached) and strands are relevant to our computation.

After a reaction has been carried out using the $\overline{xy}$ instruction strand, a complementary *release* strand, xy, is added to release the compound, and, thus, prepare for further reactions. We will not need to consider this in the algorithms, but technically, this is obtained by really using a strand $\overline{x'y'}$ for the process, where $\overline{x'y'}$ is different from $\overline{xy}$, but similar enough for the process to work, and then the release is carried out using $x'y'$, such that $\overline{x'y'}$ and $x'y'$ combine and never react with anything else again. Thus, the reason the strand $\overline{x'y'}$ must be a little different from $\overline{xy}$ is to ensure that the later release it nearly 100% effective. Similarly, if x is the tag on the resulting compound, after the release, we add the complement of y in exact matching quantity so that they will combine with all the y tags, now flowing freely in the pot, making them inert such that they can be ignored in the remaining process. This necessary use of different but similar strands further increases the need for a large edit distance between tags (when viewed as sequences) as discussed earlier. Any mismatch in quantity will result in a proportional drop in the yield.

We disallow simultaneous releases, since they lead to a low overall yield as we explain now. Releasing two compounds using $\overline{xy}$ implies that one released compound must be tagged with an x and the other with a y. Otherwise (that is, if both compounds have the same tag), we cannot control subsequent operations. But this implies the presence of free-flowing y strands and x strands from the first and second reactions, respectively. These may attach to any later $\overline{xy}$ strand, resulting in a reduced overall yield.

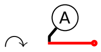
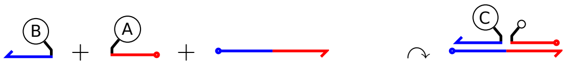
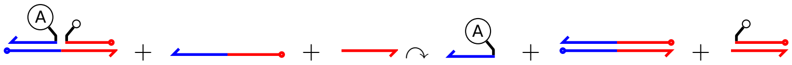
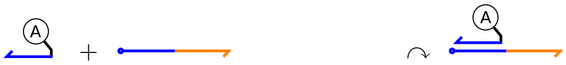
A final chemical possibility we shall use as an operation in one section is the ability to temporary *block* a compound. A compound tagged with a strand x can be blocked by adding a strand $\overline{xy}$ or $\overline{yx}$, and can be released again in the same manner as described above.

We use blocking in Sec. 3, but otherwise simply delay the release of compounds while working on others, with the aim of producing a *one-pot* program. Compounds, corresponding to the leaves, may be added gradually, but we do not allow ourselves to produce compounds corresponding to subtrees separately and add them later.

Our computational choices are the following. Given the annotated synthesis tree, we must decide on tags for the leaves and a topological ordering, including when to add, when to release, and in one algorithm also when to block and which strand to block with. Recall that given tags on the leaves, the annotation determines the tags on internal nodes. Since we most often use delayed release to avoid interference, we will frequently label internal nodes with the instruction strand, i.e., the sequence of two tags. The tag attached to the intermediate compound produced at that node is always one of the tags the strand consists of, and which one it is, is determined by the inheritance information provided by the chemist.

In summary, a program is a sequence of operations (tag, react, release, block), where *tag* attaches a specified tag to a base compound, *react* combines two intermediate compounds, *release* releases the resulting intermediate compound, and *block* blocks a compound. To be chemically feasible, left and right input compounds to any react operation must have the compound placed to the right and left, respectively, the react operations must form a topological ordering of the tree, compounds (unreleased as well as possibly blocked) must be released (unblocked) before they are used again, any block operation must use a strand

Table 1. Operations of a DNA-templated program: note, that i) the tag operation allows for attaching the compound to the left or right end of the tag, ii) the inheritance for the react operation is given as input from the chemist, iii) the release operation assumes an addition of complementary tags in order to handle waste, iv) the blocking operation can bind the tagged compound to the left or right part of the added strand.

tag	
react	
release	
block	

matching the compound tag to the left or right, all unreleased (and blocked) compounds in the pot at any given time must be unreleased (and blocked) with unique (at the time) strands, and if there are compounds in the pot with the same tag, all but one must be unreleased or blocked.

This is implied by the above, but just for emphasis, we cannot use strands of the form xx in a controlled process, so if we use τ different tags, we have at most $\tau(\tau-1)$ different strands at our disposal.

We illustrate some of these restrictions now, using the smallest possible interesting synthesis trees. First note that because compounds are at one end of a tag, we cannot have an unreleased compound with an $\overline{ab}$ strand while using ba at the root of the other subtree. This is because when we release using ab, then (without loss of generality) the released compound is tagged with a and the compound is at the right end. Thus, later, it must react with a compound tagged with a b where the compound is at the left end. Thus, the strand from that subtree would have to have the form xb for some x; see Fig. 1.

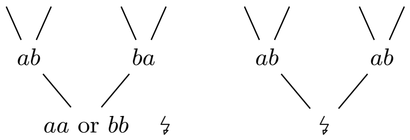


Fig. 1. Illustration of disallowed strand assignments. Left: Using strands ab and ba for two children requires the parent to be assigned either the strands aa or bb, which will result in a reduced overall yield, as with a probability of 50%, the corresponding compounds do not get in close proximity and therefore will not react. Right: Assume one subtree is already computed and the compound has to be unreleased with the complementary strand $\overline{ab}$. The corresponding unrelease needs to make the waste inert with $\overline{a}$ or $\overline{b}$, depending on which tag is now flowing freely in the pot. However, due to the disallowed simultaneous release of the other subtree, the release operation of the last of the two subtrees would accidentally make tagged compounds inert.

The reader may have wondered if the reverse sequence of x is any different from x in a pot, or if xy could interfere with yx. Starting with the latter, breaking the sequences into their nucleotides, $\alpha_1\alpha_2\cdots\alpha_{n_\alpha}\beta_1\beta_2\cdots\beta_{n_\beta}$ is different from $\beta_1\beta_2\cdots\beta_{n_\beta}\alpha_1\alpha_2\cdots\alpha_{n_\alpha}$, and they are not the reverse of each other. Obviously, x cannot be distinguished from its reverse sequence in a pot. However, compounds are attached to one of the ends, so everything has an orientation.

Finally, to give a clean initial presentation, we do not consider the option of adding multiple strands simultaneously. Computationally it does not add anything and for most problems where the objective is to use for smallest number of different strands, it is counter-productive. However, in a lab, it could be desirable to know when this is an option. One could also lift the restriction of one-pot synthesis. However, since this would lead to a multi-criteria problem, we prefer to focus on the cleaner one-pot problem.

Some of the algorithms in this paper and graphical illustrations of the chemical processes can be inspected via a prototype implementation [9].

3 Minimizing the Number of Tags

In this section, our objective is to minimize the number of tags used on base compounds (the leaves), and as our second priority, we want to minimize the total number of tags used.

It turns out that, with appropriate blocking, it is always possible to arrive at a program using only two tags on base compounds, and clearly, for any two neighboring leaves with the same parent, the tags must be different. We refer to the two tags as a and b. Using the following recursively defined function, $\left\lceil \frac{\textsc{Mnt}(\textsc{Root},0,0)}{2} \right\rceil$ will compute the minimum number of tags needed to block intermediate compounds when the basic compounds are tagged using only a and b.

Let t_a and t_b denote the subtrees of a tree t where the compound is tagged with an a and b, respectively. We keep track of tags used with a and with b separately, counting using c_a and c_b.

$$\textsc{Mnt}(t, c_a, c_b) = \begin{cases} \max(c_a, c_b) & \text{if } t \text{ is a leaf} \\ \min \left(\begin{array}{l} \max \left(\begin{array}{l} \textsc{Mnt}(t_a, c_a, c_b), \\ \textsc{Mnt}(t_b, c_a + 1, c_b) \end{array} \right), \\ \\ \max \left(\begin{array}{l} \textsc{Mnt}(t_a, c_a, c_b + 1), \\ \textsc{Mnt}(t_b, c_a, c_b) \end{array} \right) \end{array} \right) & \text{otherwise} \end{cases} \tag{1}$$

We discuss correctness and the derived program in the following (see the appendix for a simple example calculation). First, we decide arbitrarily between a and b for the final tag on the target compound that the root represents. If we use only the two tags a and b on base compounds, then we can determine all tagging recursively, since the chemist has informed us, for each node in the subtree, from which child we inherit the tag, i.e., if a node has a given tag, then a specific child of that node must have the same tag, and then the other child must be given the other tag (of the two tags a and b).

Since compounds have one of only two tags, any reaction involves both tags, so anything else in the pot must be blocked. In algorithms to be presented later, leaving them unreleased can also be an option, but in this particular case with only two tags on compounds, this would lead to the disallowed simultaneous release; see the earlier Fig. 1.

As a consequence, for any node with two non-leaf subtrees, we must decide which subtree to synthesize first, and then block while we work on the other subtree. In the subtree we synthesize first, we must block other compounds (corresponding to subtrees) recursively. We find the best subtree to block using the minimization in the formula above. The first entry in the minimization corresponds to first synthesizing and then blocking the subtree t_a. This requires no further resources while synthesizing that subtree, but while later synthesizing t_b, the compound from t_a must be blocked using a tag that has not been used for blocking subtrees on the way from the root to this node. Actually, when using some tag x to block a, for instance, this can be done (unconstrained) as ax or xa. Thus, each such tag x can be used twice, which accounts for the fraction $\frac{1}{2}$ in the final result, $\left\lceil \frac{\textsc{Mnt}(\textsc{Root},0,0)}{2} \right\rceil$.

The best values can be computed using dynamic programming. If the tree is of height h, then each of the variables a and b in the expression can take on at most h different values, so if the tree has size n, then $O(nh^2)$ is an upper bound on the number of values to be computed and each value in a given node can be computed in constant time from values in the node's subtrees, so $O(nh^2)$ is also an upper bound on the computation time. A program can easily be extracted from the computed values by simply checking if the various minima are obtained from the left or right. An example program is shown in the appendix.

4 Minimizing the Number of Strands

In this section, we consider the problem when it is undesirable to use blocking, so that is disallowed, and our objective is to minimize the number of strands used. We allow for an arbitrary number of tags. As any instruction strand requires a unique complementary release strand, they will not be counted separately. It turns out that it is necessary and sufficient to use $\mathcal{S}(t)-1$ different strands, where $\mathcal{S}(t)$ is the (Horton-)Strahler number [15] of the synthesis tree t. Referring to the previous section, where we restricted ourselves to only using two different tags on base compounds, the Strahler number many strands would not in general be sufficient. The result in this section is accomplished without using blocking.

Definition 1. *The Strahler number $\mathcal{S}(t)$ of a tree t is defined as follows: If t is a leaf, then $\mathcal{S}(t) = 1$, and if t has two subtrees t_l and t_r, then*

$$\mathcal{S}(t) = \max(\min(\mathcal{S}(t_l), \mathcal{S}(t_r)) + 1, \max(\mathcal{S}(t_l), \mathcal{S}(t_r)))$$

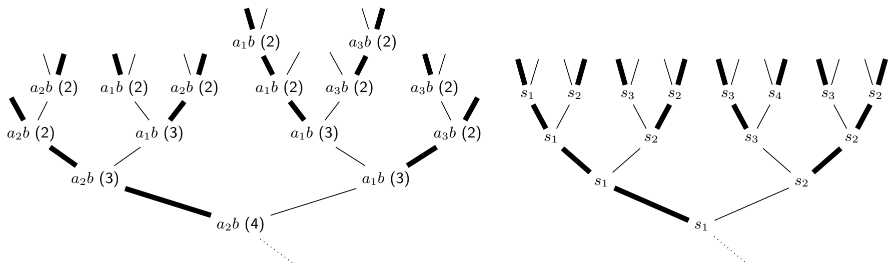


Fig. 2. Left: Illustration of the labeling algorithm that uses $\mathcal{S}(t) - 1$ many strands $a_1b, a_2b, \ldots$. Note that this is only one of the possible labelings, since strands are simply chosen from an available set, though we have consistently chosen the smallest indexed a_i available. The Strahler number is given in parenthesis. Right: Illustration of the labeling algorithm for complete binary trees: $s_1, s_2, \ldots$ is an antipath of strands, inheritance of tags is illustrated by bold lines.

$\mathcal{S}(t)$ is also referred to as the register number, i.e., the minimum number of registers required for evaluating a given arithmetic expression [6].

We are given a synthesis tree and information regarding from which child a node inherits its tag. To explain the tagging, it is easiest for us first to reorder the subtrees so that tags are inherited from subtrees according to a specific pattern. By a *layer* in a tree, we denote all the nodes of the same distance from the root. Given the synthesis tree, we order the subtrees such that when considering any layers from the left to the right, the tag is inherited alternately either from the left or from the right child, and we start by inheriting from the left; see bold edges in Fig. 2.

Now, we explain how we label each node in our synthesis tree, excluding the leaves that contain the base compounds. For the labeling, we use the set $I = \{a_1b, a_2b, a_3b, \ldots\}$. This set contains strands that have pairwise different tags as their first parts (a_i) and identical tags as their second parts (b).

Recursively for a subtree t of the synthesis tree with ordered children as described above, we first compute the subtree with the larger Strahler number. In case of identical Strahler numbers, we choose the left subtree first. The strand assignment is done as follows: In case the subtrees have identical Strahler numbers, the subtree computed first will require a strand for the release operation. This strand cannot be used for any operation in the other subtree. If the Strahler numbers are different, this constraint will not apply. However, in all cases, neighboring operations need to use different strands. During the recursion, we keep track of the set of forbidden strands (this set grows by one element for the right subtree in the case of identical Strahler numbers) and the sibling reaction strand. Note that the constraint for the sibling reaction *only* applies to the sibling reaction. The pseudo-code is given in Alg. 1 and an illustration with an example of the labeling for a tree with Strahler number 4 is given in Fig. 2 (Left). With regards to the number of strands, it is clear that the forbidden set F grows with the Strahler number, so if it was not for the temporary restriction given by the sibling, we use $\mathcal{S}(t)-1$ strands. Recall that a leaf (with a compound) has Strahler number one, so the smallest subtree we assign a strand to has Strahler number

Algorithm 1 Strahler Number Strands

```
Given: Synthesis tree $t$                    ▷ ordered children according to text description
       Set $A = \{a_1 b, a_2 b, \ldots, a_{\mathcal{S}(t)-1} b\}$    ▷ $A$: set of strands with $|A| = \mathcal{S}(t) - 1$
function AssignStrand(Tree $t$, Set $F$, Strand $sibling$)    ▷ $F$: forbidden strands
    $t_l, t_r \leftarrow$ LeftSubtree(t), RightSubtree(t)
    if both $t_l$ and $t_r$ are base compounds (leaves) then
        choose strand $s$ from $A \setminus (F \cup \{sibling\})$
    else if one of $t_l$ and $t_r$ is a base compound (a leaf) then
        $t_x \leftarrow \arg\max_{t_i \in \{t_l, t_r\}} \mathcal{S}(t_i)$    ▷ $t_x$ is the non-leaf tree
        $s \leftarrow$ AssignStrand($t_x$, $F$, _)
    else
        if $\mathcal{S}(t_l) > \mathcal{S}(t_r)$ then
            $s \leftarrow$ AssignStrand($t_l$, $F$, _)
            AssignStrand($t_r$, $F$, $s$)
        else if $\mathcal{S}(t_l) < \mathcal{S}(t_r)$ then
            $s_r \leftarrow$ AssignStrand($t_r$, $F$, _)
            $s \leftarrow$ AssignStrand($t_l$, $F$, $s_r$)
        else                                  ▷ $\mathcal{S}(t_l) = \mathcal{S}(t_r)$
            $s \leftarrow$ AssignStrand($t_l$, $F$, _)
            AssignStrand($t_r$, $F \cup \{s\}$, $s$)
    assign $s$ to $t$
    return $s$
AssignStrand($t$, $\varnothing$, _)
```

two. With regards to the restriction, when the number of available strands is at least two, the temporary restriction does not matter, since we still have a strand we can choose. Thus, the only possible problem is when we recur from a tree with Strahler number three to smaller subtrees. If the subtrees have different Strahler numbers, there is no problem, since the restriction is imposed on the smaller one. If they have the same Strahler number, the sibling restriction coincides with the growing forbidden set, so only one strand option disappears, and the one required strand can be found.

With regards to chemical feasibility, siblings have different strands by construction, and b has its compound at the left and the compound coming from the right subtree will always be tagged with b. The opposite holds for the a_is, so the strands listed in the internal nodes indicate instruction strands fulfilling all requirements.

The upper bound just given is the interesting one. The lower bound that $\mathcal{S}(t) - 1$ different strands are necessary follows directly from the equivalent result for arithmetic expressions [6]; it is simply a matter of having to store at least that many intermediate results.

Strahler examples, as the ones produced in this section, can be found in [9].

5 Complete Binary Trees

The two problems of minimizing the number of strands used (Sec. 4) and minimizing the number of tags used under the constraint that all base compounds are tagged by one out of two tags (Sec. 3) can both be solved optimally in an efficient manner. In this section, we restrict the topology of the synthesis plan to complete binary trees and present an approach that minimizes the overall number of strands *as well as* bounds the overall number of tags to the optimal, possibly plus one. We accomplish this without using blocking.

The approach will employ so-called antipaths [5], which is a sequence of adjacent edges in a digraph, where every visited edge has opposite direction of the previously visited edge; and we will need some further restrictions defined below.

Definition 2. *An antipath in a digraph is a finite sequence of edges* (u_i, u_j) *having one of the following forms:*

$$(u_1, u_2), (u_3, u_2), (u_3, u_4), \ldots \quad \textit{or} \quad (u_2, u_1), (u_2, u_3), (u_4, u_3), \ldots$$

An antipath is called return-free *if for any two successive edges* (u, v) *and* (u', v')*,* $\{u, v, \} \neq \{u', v'\}$ *and* non-overlapping *if no edge is used twice.*

In our construction, we will need return-free, non-overlapping antipaths as long as possible (each edge will correspond to a strand) from digraphs with as few vertices as possible (each vertex will correspond to a tag). The proof of the theorem below is available in the full version of our paper.

Theorem 1. *In a complete digraph* $G_n = (V, E)$ *over* $n \geq 2$ *vertices, the length of a longest return-free, non-overlapping antipath is* $n(n-1)$ *if* n *is odd and* $n(n-2)+1$ *if* n *is even.*

As in all the other sections, we are given a synthesis tree and information regarding from which child a node inherits its tag. We reorder subtrees with regards to inheritance as in the previous section.

Separate from the tree structure, assume that we let each tag that we use represent a vertex in a digraph. Thus, a directed edge in the digraph is an ordered pair of tags, which we can interpret as a strand. We choose a longest antipath $s_1, s_2, s_3, \ldots$ in such a digraph, writing them as s_i for the ith strand. The number of tags (equal to the number of vertices in the digraph) we use depends on how long an antipath we need for the construction below.

First, we explain how we label each node in our synthesis tree, excluding the leaves that contain the base compounds. The root is labeled s_1 and, for ease of the definition below, artificially assume that the root has a parent, and that we moved left to get to the root. Moving from the root towards a leaf, we label each node with the same label as its parent (below it in our illustrations) if we move in the same direction as from the grandparent to our parent, and we label it with the next label (index one larger) if we change direction; see

Fig. 2 (Right). Afterwards, the base compounds in the leaves can be tagged in the obvious manner, tagging the left (right) leaf with the left (right) part of its parent's strand.

From the labeled synthesis tree, we can define the program recursively. For a given node, we first compute the subtree whose root has the strand with the smallest index, leaving it unreleased while the other subtree is computed, after which the first subtree is released and the instruction of the node is carried out.

We now argue that the labeling algorithm produces a chemically feasible program. With regards to the reactions, due to the definition of the inheritance, a simple inductive argument establishes that at any node, the two input compounds stem from the left-most and right-most leaves of the subtree of the node. Thus, the compounds are tagged with the correct orientation for a reaction. With regards to the interference, the definition of the program explicitly states that the subtree whose root is labeled with the smallest indexed strand is computed first, and by the labeling algorithm, that strand is not used in the other subtree. Thus, no release operation can unintendedly release more than one compound.

Theorem 2. *The labeling algorithm uses the minimal number of strands and at most one more than the minimal number of tags.*

Proof. A complete binary tree of height h has Strahler number $h+1$, so we know from Section 4 that h is the optimal number of strands. The maximal number of direction changes from the root to the level next to the leaves is $h-1$, so, since the root is labeled s_1, the maximal label index is $1+(h-1)=h$.

Assume that it is somehow possible to make a program using the optimal number of tags τ. Observe that we can make at most $\tau(\tau-1)$ different strands from τ tags, so if τ is the optimal number tags, this must mean that this hypothetical program uses at most $\tau(\tau-1)$ strands.

If we allow for $\tau+1$ tags in our program, we know from Theorem 1 that an antipath of length at least $(\tau+1)(\tau-1)+1$ exists. Since we use the optimal number of strands and $(\tau+1)(\tau-1)+1 \geq \tau(\tau-1)$ for any positive integer τ, the theorem follows. □

We remark that the construction is actually optimal also with regards to the number of tags in many cases. In fact, for all heights up to 25, we know that we are optimal, except for the heights 10–12. An example argument that the method is optimal for height 13 (in fact, the same argument works up to height 20) goes as follows. We know we need 13 different strands. With 4 tags, we can make only $4 \cdot 3 = 12$ different strands, so 5 tags are necessary for any program, and with 5 tags we can find antipaths of lengths up to $5 \cdot 4 = 20$. Similarly, for height 9 (in fact, down to height 7), we need four tags to have enough strands, and with four tags, we can make antipaths of lengths up to $4 \cdot 2 + 1 = 9$. It is the slightly limited lengths of antipaths for an even number of tags that prevents us from extending this optimality argument throughout the range 10–12.

Finally, the algorithm runs in linear time. The recursive definition of the longest antipath one can extract from the theorem is constructive and easily implemented in linear time in the number of strands needed for the synthesis tree algorithm, the labeling is a linear-time pre-order traversal, and the extraction of the program is a linear-time depth-first traversal.

6 Concluding Remarks

For small synthesis trees, one might consider *all* possible programs, i.e., all topological orderings of the synthesis tree with optional blocking at any node. For all such programs, one can find an optimal assignment of tags and strands. In the full version of the paper, we specify an integer linear program that does that.

Directly related to the questions we consider, it would be interesting to settle the near-optimality issue for complete binary trees, where we have provably optimal results for heights up to 25, except for heights 10–12. It may be necessary to loosen the constraint of using antipaths for the labeling slightly, but it requires great care to still ensure correctness. Also in relation to the complete binary tree algorithm, solutions could be used as the basis for solutions for trees that are not complete. For instance, adding long paths to a complete binary tree need not result in a higher cost in terms of number of tags and strands. It seems that for trees in general, the largest induced complete binary tree is the key to the cost and a formal extension from complete binary trees to trees in general exploiting this kernelization-like idea would be nice.

A quite different direction is to explore concurrency. If one uses more tags and strands than the bare minimum, some subtrees may become independent and even one-pot synthesis could allow for concurrency. Trade-off results between concurrency maximization and tag/strand minimization would be interesting.

Acknowledgment

The second and third authors were supported in part by the Danish Council for Independent Research, Natural Sciences, grants DFF-1323-00247 and DFF-7014-00041.

References

1. L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 5187:1021–1024, 1994.
2. J. L. Andersen, C. Flamm, M. M. Hanczyc, and D. Merkle. Towards optimal DNA-templated computing. *International Journal of Unconventional Computing*, 11(3–4):185–203, 2015.
3. E. Benson, A. Mohammed, J. Gardell, S. Masich, E. Czeizler, P. Orponen, and B. Högberg. DNA rendering of polyhedral meshes at the nanoscale. *Nature*, 523:441–444, 2015.
4. L. Cardelli. Two-domain DNA strand displacement. In *DCM*, pages 47–61, 2010.
5. D. de Werra and C. Pasche. Paths, chains, and antipaths. *Networks*, 19(1):107–115, 1989.
6. P. Flajolet, J. Raoult, and J. Vuillemin. The number of registers required for evaluating arithmetic expressions. *Theoretical Computer Science*, 9(1):99–125, 1979.

7. R. A. Goodnow Jr, C. E. Dumelin, and A. D. Keefe. DNA-encoded chemistry: enabling the deeper sampling of chemical space. *Nature Reviews Drug Discovery*, 16:131–147, 2017.
8. K. Gorska and N. Winssinger. Reactions templated by nucleic acids: More ways to translate oligonucleotide-based instructions into emerging function. *Angewandte Chemie International Edition*, 52(27):6820–6843, 2013.
9. B. N. Hansen and A. Mihalchuk. DNA-Templated Computing. Master's thesis, University of Southern Denmark, Denmark, 2015. `http://cheminf.imada.sdu.dk/dna/` [Accessed March 30, 2017].
10. Y. He and D. R. Liu. A sequential strand-displacement strategy enables efficient six-step DNA-templated synthesis. *Journal of the American Chemical Society*, 133(26):9972–9975, 2011.
11. J. B. Hendrickson. Systematic synthesis design. 6. Yield analysis and convergency. *J. Am. Chem. Soc.*, 99:5439–5450, 1977.
12. X. Li and D. R. Liu. DNA-templated organic synthesis: Nature's strategy for controlling chemical reactivity applied to synthetic molecules. *Angewandte Chemie International Edition*, 43:4848–4870, 2004.
13. W. Meng, R. Muscat, M. McKee, P. M. amd A.H. El-Sagheer, J. Bath, B. Davis, T. Brown, R. O'Reilly, and A. Turberfield. An autonomous molecular assembler for programmable chemical synthesis. *Nature Chemistry*, 2016. doi:10.1038/nchem.2495.
14. A. Phillips and L. Cardelli. A programming language for composable DNA circuits. *J. R. Soc. Interface*, 6(Suppl 4):S419–S436, 2009.
15. A. Strahler. Hypsometric (area-altitude) analysis of erosional topography. *Bulletin Geological Society of America*, 63:1117–1142, 1952.
16. S. Wickham, J. Bath, Y. Katsuda, M. Endo, K. Hidaka, H. Sugiyama, and A. Turberfield. A DNA-based molecular motor that can navigate a network of tracks. *Nature Nanotechnology*, 7:169–173, 2012.

Appendix: DNA Program Example

We consider an example synthesis tree with four base compounds. The actual names of the compounds is not used in any of our algorithms, but for illustration, assume the base compounds are A, B, C, and D. Furthermore, we assume that the tagged compound A reacts with the tagged compound B ($A + B \to E$), and that E will have the tag of B. The complete assumptions are

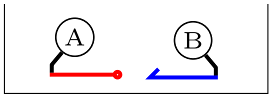
$$\begin{aligned} A + B \to E,&\ E \text{ will inherit the tag of } B\\ C + D \to F,&\ F \text{ will inherit the tag of } C\\ E + F \to X,&\ X \text{ will inherit the tag of } E \end{aligned}$$

and we demonstrate one possible program computing the target compound X as a one-pot synthesis.

We first tag the base compounds A at the left end of the tag a and B at the right end of the tag b. The tag a (respectively b) is depicted as a red (respectively blue) line in the following.

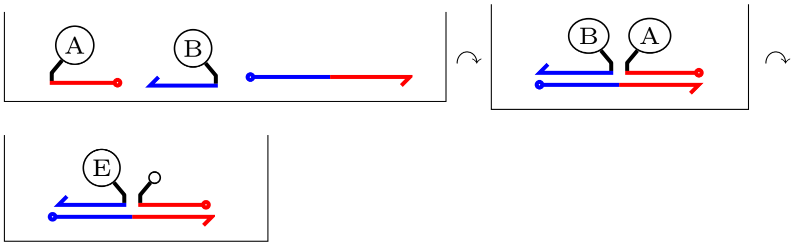
```
tag($A$, $a$, left)
tag($B$, $b$, right)
```

The state is as follows:



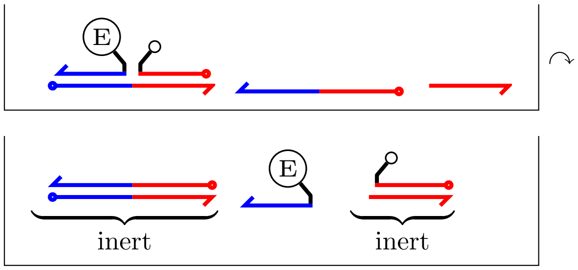
We add the complementary strand $\overline{ba}$ in order to bring A and B in close vicinity and they react to produce E. In this process, A loses its tag.

```
react($\overline{ba}$)
```



We release the produced tagged compound E with the strand ba and E is now tagged with b. The tag a is now unattached and we add the complementary tag $\overline{a}$ such that in the subsequent operations, it can be ignored.

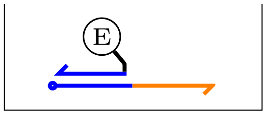
```
release($ba$)
```



Since they are no longer relevant, we will not depict the inert strands in the following.

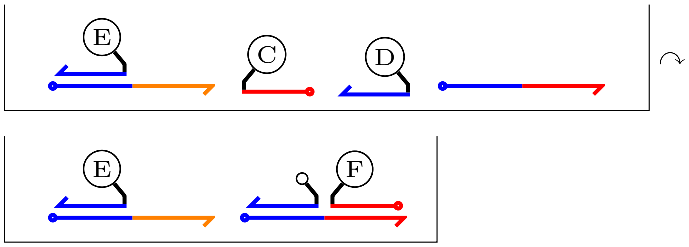
In order to avoid unintended interference, we block the tagged compound E with a strand $\overline{bc}$ (c shown in orange).

```
block($\overline{bc}$)
```



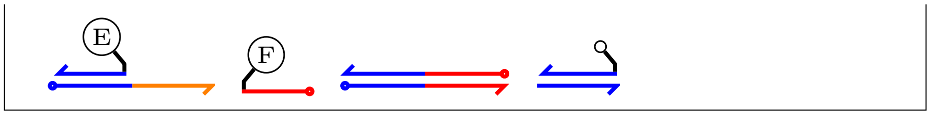
We proceed with the base compounds C and D in a similar manner. Note that C is tagged with a and D is tagged with b, i.e., adding them to the pot in the beginning would have led to unintended interference. By adding $\overline{ba}$, the tagged compounds C and D react to produce F, and D loses its tag.

```
tag(C, a, left)
tag(D, b, right)
react(b̅a̅)
```



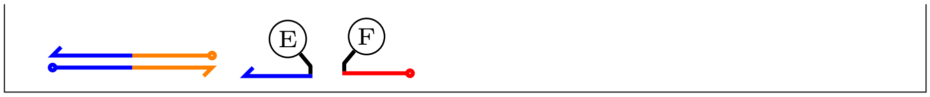
We then release the tagged compound F using the strand ba and pacify the tag b.

```
release(ba)
```



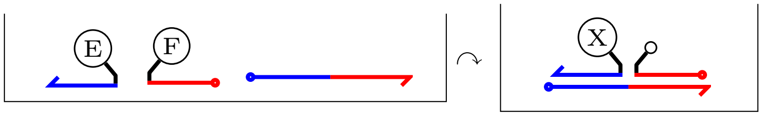
The blocked tagged compound E is released with the strand bc.

```
release(bc)
```



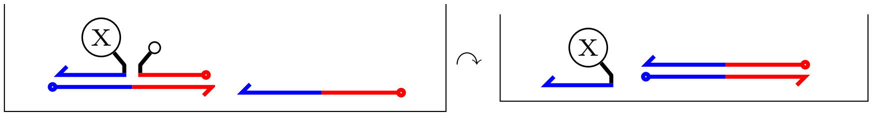
Finally, the tagged compounds E and F are brought in close vicinity using the strand $\overline{ba}$, producing X, and F loses its tag.

```
react(b̅a̅)
```



In the very last step, the target compound is released using strand ba, which finalizes the synthesis.

```
release(ba)
```



The only non-inert tag is the tag attached to compound X, which makes it chemically easy to extract the compound from the pot. The synthesis required three different tags and two different strands (and their corresponding complementary tags and strands).

The given example also illustrates the minimization of the number of tags for blocking, when assuming that only two tags on the compounds are used (see Eqn. 1), and the number of tags for blocking is to be minimized. Without loss of generality, we choose the goal compound X to be tagged with b. Given that decision, and given that we have restricted ourselves to using only two different tags on the compounds, there are no further choices for tagging: The tagging of all nodes in the tree is simply inferred as follows. The nodes A, C, and F need to be tagged with an a, and B, D, and E with a b. In this example, the subtree of the root X corresponding to $A + B \rightarrow E$ is synthesized before the subtree corresponding to $C + D \rightarrow F$. As we need to block the result of the former synthesis, we need an additional tag for blocking for the subtree E. With respect to Eqn. 1, this corresponds to the recursive calculations for the inference $\max(\textsc{Mnt}(E, 0, 0), \textsc{Mnt}(F, 1, 0))$ (the choice to synthesize the subtree $C + D \rightarrow F$ first would, in this specific example, lead to the same overall result). This leads to the following base cases for the leaves: $\textsc{Mnt}(A, 0, 0) = 0$ and $\textsc{Mnt}(B, 0, 0) = 0$, and for the other subtree $\textsc{Mnt}(C, 1, 0) = 1$ and $\textsc{Mnt}(D, 1, 0) = 1$. Obviously, $\textsc{Mnt}(E, 0, 0) = 0$ and $\textsc{Mnt}(F, 1, 0) = 1$, leading to $\textsc{Mnt}(X, 0, 0) = \min(\max(\textsc{Mnt}(E, 0, 0), \textsc{Mnt}(F, 1, 0)), \ldots) = 1$. Thus, only one additional tag is needed for blocking.