DNA-Templated Computing

Bjarke Niemann Hansen
Alexei Mihalchuk

Daniel Merkle, Kim Skak Larsen, Christoph Flamm
Our Thesis

• Given a synthesis plan, automatically infer an optimal DNA-templated program.

• Looking at one-pot synthesis from a computer science point of view.

Towards an Optimal DNA-Templated Molecular Assembler
Jakob L. Andersen, Christoph Flamm, Martin M. Hanczyc, Daniel Merkle
DNA-Templated Synthesis

Programmable One-Pot Multistep Organic Synthesis Using DNA Junctions - McKee et al. 2012
Basic Notation

- **Compound** (uppercase), **Tag** (lowercase)

- **Strand**: sequence of two tags
Synthesis Plan

• A series of reactions that produces a goal compound.

• Assumed to be a binary tree.

\[
\begin{align*}
B & \quad + \quad A \quad \rightarrow \quad E \\
C & \quad + \quad D \quad \rightarrow \quad F \\
E & \quad + \quad F \quad \rightarrow \quad X
\end{align*}
\]
Synthesis Plan
Basic Requirements

• Two compounds present in the pot at the same time must have a distinct tag.

• Two strands present in the pot at the same time must differ in at least one of their tags.
Basic Requirements

- Two compounds present in the pot at the same time must have a distinct tag.

- Two strands present in the pot at the same time must differ in at least one of their tags.
Basic Requirements

• Two compounds present in the pot at the same time must have a distinct tag.

• Two strands present in the pot at the same time must differ in at least one of their tags.
Motivating Example
Why is this hard?

- In which order to perform the reactions?
- Inheritance of tags and their position.
- Avoiding side products and unwanted bindings.
- Different optimization goals.
Optimization Goals

• *Program length*
  Fewer instructions but more distinct tags required.

• *No. of tags*
  Fewer tags, at the cost of a longer program.

• *No. of strands*
  Fewer distinct strands but not fewest tags.
Integer Linear Programming

- Inspired by liveness analysis from Compiler Theory.
- Given an arbitrary program, if two compounds or two strands are *alive* at the same time, they must be distinct.
- Solve such a program using an ILP model.
The Approach

• Enumerate all the ways to traverse a plan.
• Enumerate all the ways to construct a program.
• Determine the interference between the tags.
• Solve the interference constraints using ILP.
Interference

If two strands are alive at the same time, they should be unique.

Tags should be unique.

minimize \[ y \]

subject to \[ \sum_{c} x_{ic} = 1 \quad \forall i \in V \]

\[ x_{ic} + x_{jc} \leq 1 \quad \forall ij \in E, \forall c \in V \]

\[ x_{ic} + x_{kc} \leq 1 + z_{ijkl} \quad \forall ijk \in S, \forall c \in V \]

\[ x_{jc} + x_{lc} \leq 1 + (1 - z_{ijkl}) \quad \forall ijk \in S, \forall c \in V \]

\[ \sum_{c} cx_{ic} \leq y \quad \forall i \in V \]

\[ x_{ic} \in \{0, 1\} \quad \forall i, \forall c \in V \]

\[ y \geq 0 \]
Example
Integer Linear Programming

• An easy way to generate all (or a subset of) optimal programs for a synthesis plan.

• However, runs in exponential time and quickly becomes impractical.
<table>
<thead>
<tr>
<th>Tags on Compounds</th>
<th>Atomic</th>
<th>Toeholds</th>
<th>Fewest Tags</th>
<th>Shortest Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Yes</td>
<td>No</td>
<td>Polynomial</td>
<td>Linear</td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
<td>Yes</td>
<td>Polynomial</td>
<td>Linear</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>No</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Yes</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>Yes</td>
<td>No</td>
<td>Polynomial</td>
<td>Linear</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>Yes</td>
<td>Yes</td>
<td><em>Exponential</em></td>
<td>Linear</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>No</td>
<td>No</td>
<td><em>Exponential</em></td>
<td>Linear</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>No</td>
<td>Yes</td>
<td><em>Exponential</em></td>
<td>Linear</td>
</tr>
<tr>
<td>k</td>
<td>Yes</td>
<td>No</td>
<td>Polynomial</td>
<td><em>Exponential</em></td>
</tr>
<tr>
<td>k</td>
<td>Yes</td>
<td>Yes</td>
<td><em>Exponential</em></td>
<td><em>Exponential</em></td>
</tr>
<tr>
<td>k</td>
<td>No</td>
<td>No</td>
<td><em>Exponential</em></td>
<td><em>Exponential</em></td>
</tr>
<tr>
<td>k</td>
<td>No</td>
<td>Yes</td>
<td><em>Exponential</em></td>
<td><em>Exponential</em></td>
</tr>
</tbody>
</table>
Two Tags on Compounds

• Tagging compounds is expensive!

• Reuse the same two tags (a and b) for all the compounds.
Atomicity
Atomicity
Temporal Protection
Temporal Protection
Temporal Protection
Temporal Protection

Diagram with nodes labeled A, B, C, D, E, F, and X. Connections between nodes are indicated with arrows.
Example
The Algorithm

\[ F(node, a, b) \]

\[
\begin{cases}
\min & \left\{ \begin{array}{l}
\max & F(node.left, a, b) \\
F(node.right, a + 1, b) \\
F(node.right, a, b) \\
F(node.left, a, b + 1) \\
\end{array} \right. 
\end{cases}
\]
Two Tags on Compounds

- A program with two tags on the compounds and fewest tags overall.
- Runs in polynomial time.

\[ O(n \cdot h \cdot h) \]

nodes block a block b
<table>
<thead>
<tr>
<th>Tags on Compounds</th>
<th>Atomic</th>
<th>Toeholds</th>
<th>Fewest Tags</th>
<th>Shortest Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Yes</td>
<td>No</td>
<td>Polynomial</td>
<td>Linear</td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
<td>Yes</td>
<td>Polynomial</td>
<td>Linear</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>No</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Yes</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>Yes</td>
<td>No</td>
<td>Polynomial</td>
<td>Linear</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>Yes</td>
<td>Yes</td>
<td><em>Exponential</em></td>
<td>Linear</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>No</td>
<td>No</td>
<td><em>Exponential</em></td>
<td>Linear</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>No</td>
<td>Yes</td>
<td><em>Exponential</em></td>
<td>Linear</td>
</tr>
<tr>
<td>k</td>
<td>Yes</td>
<td>No</td>
<td>Polynomial</td>
<td><em>Exponential</em></td>
</tr>
<tr>
<td>k</td>
<td>Yes</td>
<td>Yes</td>
<td><em>Exponential</em></td>
<td><em>Exponential</em></td>
</tr>
<tr>
<td>k</td>
<td>No</td>
<td>No</td>
<td><em>Exponential</em></td>
<td><em>Exponential</em></td>
</tr>
<tr>
<td>k</td>
<td>No</td>
<td>Yes</td>
<td><em>Exponential</em></td>
<td><em>Exponential</em></td>
</tr>
</tbody>
</table>
Fewest Tags Without Atomicity

- Atomicity is required for Two Tags on Compounds.
- Let's remove the requirement!
Improved Protection
Example
The Algorithm

\[ F(\text{node, strand, } S) \]

\[
\min \left\{ \begin{array}{l}
\max \left\{ \begin{array}{ll}
F(\text{node.left, } xy, S) & \forall xy \\
F(\text{node.right, } zw, S \cup \{xy\}) & \forall zw \\
F(\text{node.right, } zw, S) & \forall zw \\
F(\text{node.left, } xy, S \cup \{zw\}) & \forall xy
\end{array} \right. \\
\end{array} \right. \]

Diagram:
The Algorithm

• A program with fewest tags overall.

• Runs in exponential time.

\[ O(n \cdot \log^2 n \cdot 2^{\log^2 n} \cdot \log^2 n) \]

- nodes
- strands
- blocked
- inner work
The Algorithm

• A program with fewest tags overall.

• Runs in exponential time.

\[ O(n \cdot \log^2 n \cdot 2^{\log^2 n} \cdot \log^2 n) \]

\[ \text{nodes} \quad \text{strands} \quad \text{blocked} \quad \text{inner work} \]
Empirical Analysis

- Assume a program with 4 tags and 5 strands that is solving a synthesis plan.

- A solution can be seen as a set of strands.
Strand Sets

• A solution can be seen as the set of strands used.

• Unlabelled connected loop-free digraphs with \#tags nodes and \#strands edges.

http://oeis.org/A052283
Strand Sets

- The number of strands that can be made with \( t \) tags is
  \[ |S| = t(t - 1) \]
- The number of tags needed to create \( |S| \) strands is then
  \[ t \in O\left(\sqrt{|S|}\right) \]
Complete Binary Trees

• Can be solved with $\log(n)$ strands.

• The strands are selected consecutively.

• The resulting program has an optimal number of strands and an optimal number of tags.
General Binary Trees

- Fix strand set to $S = \{aX, bX, cX, dX, \ldots\}$, $|S| = \log(n)$
- Register allocation in Expression Trees.
- Shows that upper bound is $\log(n) + 1$ tags.
- Approach optimal with regard to no. of strands, but not necessarily with regard to no. of tags.
Comparison

4 strands
3 tags

3 strands
4 tags
Going Polynomial

• Brute force approach.

\[
\text{for } t = \sqrt{\log n \ldots \log n + 1} \text{ do}
\]
\[
\text{for } |S| = \log n \ldots t(t - 1) \text{ do}
\]
\[
\quad \text{sets} = \text{select every combination from all } \binom{t(t-1)}{|S|} \text{ possible}
\]
\[
\quad \text{for } S \in \text{sets} \text{ do}
\]
\[
\quad \text{try to find solution using } S
\]
\[
\quad \text{end for}
\]
\[
\text{end for}
\]
\[
\text{end for}
\]
Going Polynomial

• Optimistic brute force approach, based on empirical analysis.

\[
\text{for } t = \sqrt{\log n} \ldots \sqrt{\log n} + 1 \text{ do}
\]

\[
\text{for } |S| = \log n \ldots t(t - 1) \text{ do}
\]

\[
\text{sets } = \text{select every combination from all } \binom{t(t-1)}{|S|} \text{ possible}
\]

\[
\text{for } S \in \text{sets} \text{ do}
\]

\[
\text{try to find solution using } S
\]

\[
\text{end for}
\]

\[
\text{end for}
\]

\[
\text{end for}
\]
Going Polynomial

- Optimistic brute force approach, based on empirical analysis.

\[
\text{for } t = \sqrt{\log n} \ldots \sqrt{\log n} + 1 \text{ do}
\]\[
\text{for } |S| = \log n \ldots t(t - 1) \text{ do}
\]

\[
\text{sets } = \text{select every combination from all } \binom{t(t-1)}{|S|} \text{ possible}
\]

\[
\text{for } S \in \text{sets} \text{ do}
\]

\[
\text{try to find solution using } S
\]

\[
\text{end for}
\]

\[
\text{end for}
\]

\[
\text{end for}
\]
Conclusion

• An ILP formulation for determining tags for programs.

• Known to be polynomial time:
  • Programs of optimal length.
  • Programs with two tags on compounds.
  • Programs with fewest tags (for complete binary trees).
  • Programs with fewest strands.

• "Empirically polynomial":
  • Programs with fewest tags.