Multiplex Dispensation Order Generation for Pyrosequencing

Mats Carlsson\textsuperscript{1} and Nicolas Beldiceanu\textsuperscript{2,3}

\textsuperscript{1} SICS, P.O. Box 1263, SE-164 29 KISTA, Sweden
matsc@sics.se

\textsuperscript{2} LINA FRE CNRS 2729
École des Mines de Nantes
FR-44307 NANTES Cedex 3, France
Nicolas.Beldiceanu@emn.fr

\textsuperscript{3} This research was carried out while N. Beldiceanu was at SICS.

Abstract. This article introduces the multiplex dispensation order generation problem, a real-life combinatorial problem that arises in the context of analyzing large numbers of short to medium length DNA sequences. The problem is modeled as a constraint optimization problem (COP). We present the COP, its constraint programming formulation, and a custom search procedure. We give some experimental data supporting our design decisions. One of the lessons learnt from this study is that the ease with which the relevant constraints are expressed can be a crucial factor in making design decisions in the COP model.

1 Introduction

This article introduces the multiplex dispensation order generation problem, a real-life combinatorial problem that arises in the context of analyzing large numbers of short to medium length DNA sequences with the Pyrosequencing method [1–3]. DNA, or deoxyribonucleic acid, is the molecule encoding the genetic information of all cellular life. It is a double-stranded polymer formed from four nucleotides (bases) abbreviated A, C, G, T.

The Pyrosequencing method was developed based on the combination of four enzymes: (i) polymerase (bacterial), (ii) sulphurylase (yeast), (iii) luciferase (firefly) and (iv) apyrase (potato). The idea is to use a single, \emph{input} strand of the DNA to be analyzed as a template for synthesizing the complementary, \emph{output} strand. The synthesis proceeds by incorporating one nucleotide at a time, the incorporation of a specific nucleotide being quantitatively detectable as visible light; see Fig. 1.

Assuming a sample from a monoploid species, i.e. individuals have a single copy of each gene, each cycle of the method proceeds as follows\textsuperscript{1}: A specific nucleotide is dispensed, i.e. added to the reaction, and if it matches the current base of the input strand, it is incorporated into the output strand, and the next input base becomes current. If it matches a stretch of $k$ bases, it is incorporated $k$ times, and the $k^{th}$ next input base

\textsuperscript{1} We ignore details of the chemistry and the issue of complementary DNA bases.
Fig. 1. The Pyrosequencing method. A sample is analyzed wrt. a cyclic dispensation order AGTCAGTC... Assuming a monoploid sample, the sequence CTGCTTAA is obtained.

becomes current. Each incorporation event is accompanied by emission of visible light with energy proportional to the amount of incorporated nucleotide, i.e. to \( k \), the number of matched bases. No matter whether the nucleotide was incorporated, the enzymes ensure that any residue of it is degraded, and the reaction is ready for the next cycle. A histogram with one peak of height \( k \) per cycle captures the outcome of the reaction.

Fig. 1 suggests that the nucleotides should be dispensed in a cyclic order AGTCAGTC... However, in a typical application such as forensic analysis, most of the sequence is known in advance, and only single positions or small stretches are variable. It is these polymorphic parts that are of interest, i.e. that carry signal. For example, in single nucleotides polymorphism (SNP) analysis, a sequence has a single variable position. In our work, a sequence is allowed to have more general variable parts and is described by a template. A template is simply a restricted form of regular expression where \( n \) denotes a single nucleotide (\( n \in \{A, C, G, T\} \)), \( xy \) denotes concatenation, \( x/y \) denotes alternatives, \( \epsilon \) denotes the empty string, and \([x]\) is the same as \(x/\epsilon\), assuming that \(x\) and \(y\) are themselves templates. Parentheses are used for grouping. An allele of a template \( s \) is a string over the nucleotides described by \( s \).

While a cyclic dispensation order is appropriate for completely unknown sequences, it is very wasteful in terms of reagents and throughput for typical templates. Hence, an algorithm that can generate a custom, simplex dispensation order (SDO) for a specific template is of great practical value. Such an algorithm was the subject matter of a previous paper [4].

In many applications of the method, the DNA sample is from a diploid species, i.e. one where each individual has inherited one copy of each gene from the father and one from the mother. This means that the sample contains a multiset of two alleles, which may or may not be distinct. If they are distinct, the histogram obtained will be equivalent to superimposing the histograms for the two alleles. The requirement that an SDO should allow for determining the genotype of the input sample is a major challenge for the SDO generation algorithm [4], but is not discussed further in this article. Ex. 1 shows an input template and a dispensation order computed by the SDO generation
algorithm. Such dispensation orders drive the analysis equipment, which can perform some number (typically 96) of reactions in parallel.

**Example 1.** Suppose that we are given the following input template, describing a piece of DNA from a diploid species:

\[
\begin{align*}
\text{CA} & \text{ (A/C)} \text{[AGA]} \text{[TG]} \text{[A/G]} \text{TATTC}
\end{align*}
\]

The template contains 4 polymorphisms generating 24 alleles, shown in Fig. 2. A dispensation order for this template is:

\[
\text{CACAGATGATATC}
\]

<table>
<thead>
<tr>
<th>Template</th>
<th>CA A/C [AGA] [TG] [A/G] TATTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allele 1</td>
<td>CA A ε ε ε TATTC</td>
</tr>
<tr>
<td>Allele 2</td>
<td>CA A ε ε A TATTC</td>
</tr>
<tr>
<td>Allele 3</td>
<td>CA A ε ε G TATTC</td>
</tr>
<tr>
<td>Allele 4</td>
<td>CA A ε TG ε TATTC</td>
</tr>
<tr>
<td>Allele 5</td>
<td>CA A ε TG A TATTC</td>
</tr>
<tr>
<td>Allele 6</td>
<td>CA A ε TG G TATTC</td>
</tr>
<tr>
<td>Allele 7</td>
<td>CA A AGA ε ε TATTC</td>
</tr>
<tr>
<td>Allele 8</td>
<td>CA A AGA ε A TATTC</td>
</tr>
<tr>
<td>Allele 9</td>
<td>CA A AGA ε G TATTC</td>
</tr>
<tr>
<td>Allele 10</td>
<td>CA A AGA TG ε TATTC</td>
</tr>
<tr>
<td>Allele 11</td>
<td>CA A AGA TG A TATTC</td>
</tr>
<tr>
<td>Allele 12</td>
<td>CA A AGA TG G TATTC</td>
</tr>
<tr>
<td>Allele 13</td>
<td>CA C ε ε ε TATTC</td>
</tr>
<tr>
<td>Allele 14</td>
<td>CA C ε ε A TATTC</td>
</tr>
<tr>
<td>Allele 15</td>
<td>CA C ε ε G TATTC</td>
</tr>
<tr>
<td>Allele 16</td>
<td>CA C ε TG ε TATTC</td>
</tr>
<tr>
<td>Allele 17</td>
<td>CA C ε TG A TATTC</td>
</tr>
<tr>
<td>Allele 18</td>
<td>CA C ε TG G TATTC</td>
</tr>
<tr>
<td>Allele 19</td>
<td>CA C AGA ε ε TATTC</td>
</tr>
<tr>
<td>Allele 20</td>
<td>CA C AGA ε A TATTC</td>
</tr>
<tr>
<td>Allele 21</td>
<td>CA C AGA ε G TATTC</td>
</tr>
<tr>
<td>Allele 22</td>
<td>CA C AGA TG ε TATTC</td>
</tr>
<tr>
<td>Allele 23</td>
<td>CA C AGA TG A TATTC</td>
</tr>
<tr>
<td>Allele 24</td>
<td>CA C AGA TG G TATTC</td>
</tr>
</tbody>
</table>

Suppose that the analysis of a DNA sample from an individual of this species using this dispensation order yields the histogram shown in Fig. 3. From a case analysis of the possible combinations, we can deduce the genotype of the individual, namely:
and the putative sequences (there are two feasible allele pairs, 12+13 and 7+18):

<table>
<thead>
<tr>
<th>Polymorphism</th>
<th>Parent 1</th>
<th>Parent 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A/C</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>AGA/ε</td>
<td>AGA</td>
<td>ε</td>
</tr>
<tr>
<td>TG/ε</td>
<td>TG</td>
<td>ε</td>
</tr>
<tr>
<td>A/G/ε</td>
<td>G</td>
<td>ε</td>
</tr>
</tbody>
</table>

Note that, for each polymorphism, we can only deduce the multiset of variants present in the sample, i.e. we cannot unambiguously determine which parent contributed which variant. For the practical applications such as SNP analysis, it suffices to determine the multisets.

In this article, we take another step: in a typical laboratory setting, there is a large number of samples and different templates to analyze. If templates are different enough, the amount of parallelism can be increased by multiplexing some reactions. This is done by placing the samples in the same well, and analyzing the mixture with a *multiplex*
dispensation order (MDO). The point is that the length of an MDO can be much shorter than the total length of its constituent SDOs. Moreover, an MDO does not normally span the entire SDOs: it suffices to span the polymorphic parts plus a little more, to be made precise in Sect. 2.

A problem instance is defined by (i) a number of SDOs, and (ii) for each SDO, some first and follow facts computed by the SDO generation algorithm; these are explained in Sect. 3.

A dispensation order (SDO or MDO) is a sequence of items, each defined by (i) its nucleotide, (ii) its minimal multiplicity (the minimal height of the peak that this item in the sequence will yield), and (iii) its maximal multiplicity (the maximal height of the peak that this item in the sequence will yield). If the minimal and maximal multiplicities are equal, we say that we have a fixed item, otherwise we have a variable item.

For yielding a correct analysis result, it is crucial that the height of every peak corresponding to a variable item be precisely determined. Due to details of the sample preparation procedures, the concentration of DNA may vary a lot from sample to sample. This means that the signal strength, i.e. the height of each peak, is only comparable within a single (simplex) reaction, and not across reactions. This also means that there is no universal height unit; peak heights are always relative within one reaction. In order to calibrate the peak heights, the MDO generation algorithm must ensure to include at least one (preferably more than one) item of fixed and low multiplicity, so called norm-items, from each SDO.

Two SDO items can be coalesced into a single MDO item if they have the same nucleotide. That is, the peaks that the SDO items would yield become superimposed in the peak yielded by the coalesced MDO item. The more items are coalesced, the shorter the resulting MDO becomes. Fixed items may be coalesced arbitrarily. No two variable items can ever be coalesced, as doing so would confuse two signals. A variable item should preferably not be coalesced with a fixed item, as doing so compromises the accuracy of measuring its peak height. The COP model allows it, at a penalty.

Example 2. In this example, we show three input templates, their corresponding SDOs, and finally how these SDOs align into an optimal MDO. The resulting MDO will yield 6 peaks corresponding to variable items. The height of these peaks will allow for determining the exact allele composition of the input sample. The MDO also contains 7 norm-items out of 10 fixed items, yielding 10 peaks of fixed height. 3 of the MDO items were the result of coalescing SDO items. Finally, the MDO did not have to span to the end of any of the SDOs.

<table>
<thead>
<tr>
<th>Template</th>
<th>SDO</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCC (C/T) GGGAAATAATC</td>
<td>TCTGATATC</td>
</tr>
<tr>
<td>AT (C/T) CAGGGGTGCTT</td>
<td>ATCAGTGCT</td>
</tr>
<tr>
<td>GCTTCA (A/G) TGGA</td>
<td>GTCAGTGA</td>
</tr>
</tbody>
</table>
Legend:
\[\begin{array}{ll}
\text{v} & \text{Variable item.} \\
\text{n} & \text{Fixed item, norm-item.} \\
\text{f} & \text{Fixed item, not a norm-item.} \\
: & \text{Cut-off position (L in the constraint optimization model)}
\end{array}\]

The rest of the paper is organized as follows: in Sect. 2, the MDO problem is defined in more details; in Sect. 3, we present a COP model; in Sect. 4, we describe a custom search procedure; and in Sect. 5, we show a brief experimental evaluation of some design choices. Finally, we conclude.

## 2 Multiplex Dispensation Orders

An MDO is optimal if it fulfills Requirements 1–5 and has minimal cost according to Requirements 6–8. Some of these requirements, and their details, may seem arbitrary, but their purpose is to reduce the risk of an incorrect interpretation of the resulting histogram.

#### Definition 1. An item yields a norm-item if and only if it is fixed, of multiplicity at most 3, and non-coalesced.

#### Requirement 1. The MDO must span at least 4 fixed items of each SDO.

#### Requirement 2. The MDO must span at least 1 fixed item after the last variable item of each SDO.

#### Requirement 3. The MDO must span at least 1 norm-item of multiplicity 1 of each SDO.

#### Requirement 4. The MDO must span at least 1 norm-item within 10 dispensations from each variable item.

#### Requirement 5. No variable item may be coalesced with another variable item.

#### Requirement 6. For each SDO, the MDO spanning less than 2 norm-items incurs a penalty of 3.

#### Requirement 7. A fixed item coalesced with a variable item incurs a penalty of 3.

#### Requirement 8. The MDO incurs a penalty equal to its length.
3 A Constraint Optimization Model

3.1 Introduction

In this section, we will develop a COP model of the multiplex dispensation order generation problem. In the rest of this section, we will describe SDOs in more detail, some parameters which can be derived from the SDOs, the constraint variables, the cost function, and the constraints. Finally, we will describe the search procedure.

3.2 Problem Parameters

We need the following notation. \((s, i)\) denotes an item with attributes:

- \(s\), the SDO in which it occurs
- \(i\), the position in \(s\) at which it occurs,
- \(\text{nuc}(s, i)\), its nucleotide,
- \(\min(s, i) \geq 0\), its minimal multiplicity,
- \(\max(s, i) > 0\), its maximal multiplicity,
- \(\text{fix}(s, i)\) holds if and only if \(\min(s, i) = \max(s, i)\).
- \(\text{first}(s, i)\) holds if and only if there is an allele of \(s\) such that \((s, i)\) is the first item yielding a nonzero peak.

For items \((s, i)\) and \((s, j)\), we have:

- \(\text{follow}(s, i, j)\) holds if and only if there is an allele of \(s\) such that \((s, j)\) is the first item yielding a nonzero peak after item \((s, i)\).
- \(\text{nuc}(s, i) \neq \text{nuc}(s, j)\) if \(\text{follow}(s, i, j)\) holds.

3.3 Problem Variables

Each position in the MDO is called a cycle, numbered from 1 upward. For each SDO, a cycle has to be assigned to each item from the leftmost position up to the earliest cut-off point \(L\) admitted by the constraints.

- \(F \geq 0\) is the value of the cost function.
- \(L \geq 1\) is the last dispensation cycle.
- \(C_{(s, i)} \geq 1\) is the cycle that is assigned to item \((s, i)\).
- \(D^n_k \geq 1\) is the cycle in which the \(k^{th}\) dispensation of nucleotide \(n \in \{A, C, G, T\}\) occurs. For convenience, we assume that \(D^0_k = 0\). The solution, i.e. the actual MDO, can be obtained by sorting the \(D^n_k\) by ascending value, and reading the nucleotide sequence.
- \(E_{(s, i), (t, j)}\) is 1 if \((s, i)\) and \((t, j)\) are coalesced, i.e. \(C_{(s, i)} = C_{(t, j)}\), and 0 otherwise, where \(s \neq t\).
- \(N_{(s, i)}\) is 1 if item \((s, i)\) yields a norm-peak, and 0 otherwise (see below).
3.4 Cost Function

The cost function captures Requirements 6–8.

\[ L^+ \]

\[ L^+ = \sum \left\{ 3E_{(s,i)(t,j)} \mid \neg \text{fix}(s, i) \lor \neg \text{fix}(t, j) \right\} + \]

\[ \sum_s \begin{cases} 3, \text{ if } \sum_i \{N_{(s,i)}\} = 1 \\ 0, \text{ otherwise} \end{cases} \]

3.5 Problem Constraints

*Constraints intrinsic to Pyrosequencing.*

1. Only items with the same nucleotide can be coalesced.

\[ \text{nuc}(s, i) \neq \text{nuc}(t, j) \Rightarrow C_{(s,i)} \neq C_{(t,j)} \] (1)

2. The items 1, \ldots, n of each SDO must be assigned in ascending order:

\[ C_{(s,1)} < C_{(s,2)} < \cdots, \text{ for each SDO } s \] (2)

3. A first\((s, j)\) fact expresses the constraint that \(C_{(s,j)}\) be the very first dispensation of \(\text{nuc}(s, j)\). Similarly, a follow\((s, i, j)\) fact expresses the constraint that \(C_{(s,j)}\) be the first dispensation of \(\text{nuc}(s, j)\) after cycle \(C_{(s,i)}\). These constraints model the Pyrosequencing reaction, prevent invalid multiplex dispensation orders, and form the link between the \(C_{(s,i)}\) and the \(D^n_k\) variables.

\[ \text{first}(s, j) \Rightarrow C_{(s,j)} = D^n_{1, \text{nuc}(s,j)} \] (3)

\[ \text{follow}(s, i, j) \Rightarrow \exists k \mid D^n_{k-1, \text{nuc}(s,j)} < C_{(s,i)} < C_{(s,j)} = D^n_{k, \text{nuc}(s,j)} \] (4)

4. A nucleotide cannot be dispensed twice in a row. (It could, but it would be pointless, as the second dispensation would always yield a zero peak.)

\[ D^n_1 + 2 \leq D^n_2 \land D^n_2 + 2 \leq D^n_3 \land \cdots, \text{ for each } n \in \{A, C, G, T\} \] (5)

5. All dispensations occur in distinct cycles.

\[ \text{all}_c\text{distinct}(\{D^n_k\}) \] (6)

6. Dispensing past the end of any SDO would cause the reaction to fall off the piece of DNA being analyzed, yielding nonsensical output.

\[ L \leq C_{(s,i)}, \text{ for each SDO } s \text{ with final item } i \] (7)
Definition of norm-item.

\[
N_{(s,i)} = \begin{cases} 
1, & \text{if } C_{(s,i)} \leq L \land \text{fix}(s,i) \land \max(s,i) \leq 3 \land \sum_{t,j} E_{(s,i)(t,j)} = 0 \\
0, & \text{otherwise} 
\end{cases}
\]  

(8)

Requirement 1.

\[L \geq C_{(s,i)}, \text{ for each SDO } s \text{ with 4th fixed item } i\]  

(9)

Requirement 2.

\[L > C_{(s,i)}, \text{ for each SDO } s \text{ with last variable item } i\]  

(10)

Requirement 3.

\[\exists i \mid N_{(s,i)} = 1 \land \max(s,i) = 1, \text{ for each SDO } s\]  

(11)

Requirement 4.

\[\exists i \mid N_{(s,i)} = 1 \land |C_{(s,i)} - C_{(s,j)}| \leq 10, \text{ for each variable item } (s,j)\]  

(12)

Requirement 5. The following two constraints are equivalent, but posting both of them improves constraint propagation:

\[\text{all\_distinct} (\{C_{(s,i)} \mid \neg \text{fix}(s,i)\})\]  

(13)

\[\sum \{E_{(s,i)(t,j)} \mid \neg \text{fix}(s,i) \land \neg \text{fix}(t,j)\} = 0\]  

(14)

3.6 An Earlier Model

Our choice of problem variables is far from obvious, and was not the first one that we came up with. The \(C_{(s,i)}\) variables are indispensable, as they are crucial for several of the constraints. The \(D^k_c\) variables, however, are not: the solution, i.e. the actual MDO, can be obtained by sorting the \(C_{(s,i)}\) by ascending value, removing duplicates, and reading the nucleotide sequence. However, it is awkward at best to express the intrinsic constraints shown in (3) and (4) in terms of \(C_{(s,i)}\) variables only.

In an earlier model, we had:

- \(D^k_c\) variables replaced by \(K_c \in \{A, C, G, T\}\) representing the nucleotide to dispense in the \(c^{th}\) cycle, directly encoding the solution.
- (3–6) replaced by:

\[K_{C_{(s,i)}} = \text{nuc}(s,i), \text{ for each item } (s,i)\]  

(15)

\[\text{first}(s,j) \Rightarrow (\forall p \geq 1 : p < C_{(s,j)} \Rightarrow K_p \neq \text{nuc}(s,j))\]  

(16)

\[\text{follow}(s,i,j) \Rightarrow (\forall p \geq 1 : C_{(s,i)} \leq p < C_{(s,j)} \Rightarrow K_p \neq \text{nuc}(s,j))\]  

(17)

\[\forall i \geq 1 : K_i \neq K_{i+1}\]  

(18)
The $K_c$ variables have much smaller domains than the $D_{nk}$ variables, and so a model using the former would suggest a much smaller search space in theory. However, with this formulation too, (16) and (17) are somewhat awkward to express and would probably require to be implemented as custom global constraints for efficiency. This fact prompted us to come up with the model using $D_{nk}$ variables.

### 3.7 Implementation Details

The model was implemented using the CLP(FD) library of SICStus Prolog [5]. Most of the constraints in Sect. 3.5 have straightforward CLP(FD) formulations, but the following points are worth noting:

1. Reified arithmetic constraints come in very handy in linking the $E_{(s,i)(t,j)}$ and $N_{(s,i)}$ variables. A reified constraint has the form $c \equiv b$, where $c$ is any constraint and $b$ is a 0/1 variable. It expresses the fact that the value of $b$ (1 for true, 0 for false) reflects the truth value of $c$.
2. An “at least one” constraint (existential quantifiers in (11) and (12)) is conveniently expressed using an `element/3` constraint. Suppose $Cs$ is the list of $C_{(s,i)}$ variables for given $s$, and $Xs$ is a corresponding list of 0/1 variables, each reflecting the value $N_{(s,i)} = 1 \land \max(s, i) = 1$. Then the two CLP(FD) constraints:
   
   \[
   \text{element}(I, Xs, 1), \\
   \text{element}(I, Cs, Ci)
   \]

   hold with $Ci = C_{(s,i)}$ if there is an norm-item $(s, i)$ satisfying Requirement 3.
3. Similarly, the existential quantifier in (4) is expressed as follows, where $C_i$ is $C_{(s,i)}$, $C_j$ is $C_{(s,j)}$, and $Ds$ is the list of $D_{nk}$ variables. Note the use of $[-1 | Ds]$ in the third line, which allows for accessing the domain variable $D_{nk}^{muc}$ using the index $K = k$. Note also that the condition $C_{(s,i)} < C_{(s,j)}$ has already been captured by (2), and so is not re-expressed here:
   
   \[
   \text{Ch} \#< Ci, \\
   \text{element}(K, Ds, Cj), \\
   \text{element}(K, [-1 | Ds], Ch)
   \]

### 4 Search Procedure

We now present the main features of our search procedure, beginning with the task of finding feasible solutions, and moving on to the task of finding optimal solutions. In Sect. 5, we evaluate our design decisions wrt. some experimental data.

#### 4.1 Feasible Solutions

As usual in constraint programming, the search for a feasible solution is done by depth-first search: each node (choicepoint) in the search tree corresponds to a partially constructed MDO where a specific nucleotide $n$ is being chosen. Each daughter node corresponds to a specific choice of $n$. Whenever the constraint solver detects an infeasibility, the current branch is abandoned and the search backtracks to the most recent choice-point. We now describe the specifics of the search.
Variable choice. We use a cycle-based variable order, building the MDO sequentially from left to right, and stopping when a feasible value for $L$ has been reached. As no problem variables directly correspond to the MDO, we instead assign the $D^n_k$ variables, in each cycle advancing one of the $D^n$ arrays. The relevant $C_{(s,i)}$ variables get assigned by constraint propagation. In other words, the assignment effectively simulates the reaction, which in reality proceeds from left to right.

Value choice. In each cycle we have a choice between at most four nucleotides. The choice is guided by looking ahead in the SDOs. Let $i_s'$ and $i_s''$ be the next two unassigned items in SDO $s$. For each nucleotide $n \in \{A, C, G, T\}$ we compute:

$$R_1(n) = \sum_s \begin{cases} 1, & \text{if nuc}(s, i_s') = n \\ 0, & \text{otherwise} \end{cases}$$

$$R_2(n) = \sum_s \begin{cases} 1, & \text{if nuc}(s, i_s'') = n \\ 0, & \text{otherwise} \end{cases}$$

If $R_1(n) > 0$, then $n$ is a candidate nucleotide for the next cycle. The candidates are tried in descending order of $R_1(n) - R_2(n)$, using a least recently used rule to break ties.

4.2 Optimal Solutions

Let “feasible” be a procedure implementing the search for a feasible solution, with or without a cost bound. It is assumed to compute the cost of that solution. The standard constraint programming approach to optimization consists in the iterated search for feasible solutions with successively tighter and tighter cost bound; see Alg. 1. We used this scheme with two modifications, detailed below.

Binary search. Instead of successively tightening the cost bound, we narrowed the feasible range of the cost in a binary search fashion until it becomes fixed; see Alg. 2. This was found to sometimes lead to fewer iterations and faster convergence.

Oracles. Since our variable choice is static and our value choice is independent of the cost function and its bounds, we can follow the approach of Van Hentenryck and Le Provost [6]. In each iteration of binary search with a tighter upper bound, we use an oracle, or computation path. The oracle mentions, for each cycle, the nucleotide that was assigned to that cycle in the best solution so far, and any untried choices. The oracle remains valid as long as we follow its advice. On backtracking to an untried choice, the oracle is no longer valid. The advantage of using oracles is twofold:

– The oracle guides the search so as to avoid parts of the search space that have already been exhausted.
– As long as we follow the oracle’s advice, i.e. up to backtracking, we avoid the cost of computing a value order for the current cycle.

The expression $\min(F)$ yields the current cost lower bound.
The pseudocode for our full search procedure is shown in Alg. 3. Here, the procedure “feasible” has been augmented with an input oracle $O$, which remains valid while the leftmost branch is taken in all choices. Also, “feasible” is assumed to compute an output oracle encoding the part of the search space that is still unexplored.

**PROCEDURE** label($F$) : (false, true)
Ensure: $F$ is an output parameter.
1: if feasible = true then
2: $F$ ← the current cost value
3: display data about the solution found
4: return true
5: else
6: return false
7: end if

**PROCEDURE** opt($F$) : (false, true)
Ensure: $F$ is an output parameter, receiving the optimal cost value on success.
1: if label($F_0$) = false then
2: return false
3: end if
4: while min($F$) < $F_0$ do
5: if post $F$ < $F_0$ ∧ label($F_1$) = true then
6: $F_0$ ← $F_1$
7: else
8: $F$ ← $F_0$
9: end if
10: end while
11: return true

**Algorithm 1:** Optimization

5 Performance Evaluation

In this section, we provide some experimental data supporting our design choices. The data is presented in four scatter plots (see Fig. 4), each comparing the implemented, baseline solution with a mutated version in which some feature has been changed or disabled. Each scatter plots shows some points and an $x = y$ line. The X and Y coordinates of each point give the runtime in milliseconds of a given problem instance for the baseline and mutated version, respectively.

In the evaluation, we used 184 random test instances, each consisting of three multiplexed templates, generated as follows:

- Each test instance consisted of three templates to be multiplexed.
- Each template was 12 nucleotides long and contained at least one polymorphic nucleotide. Nucleotides were polymorphic with probability $1/12$.
- Each test instance was feasible and its optimal solution was found in less than one minute.
PROCEDURE labelbin($F$) : (false, true)
Ensure: $F$ is an output parameter.
1: if feasible = true then
2: $F$ ← the current cost value
3: display data about the solution found
4: return true
5: else
6: return false
7: end if

PROCEDURE optbin($F$) : (false, true)
Ensure: $F$ is an output parameter, receiving the optimal cost value on success.
1: if labelbin($F_0$) = false then
2: return false
3: end if
4: while min($F$) < $F_0$ do
5: if post $F \leq (\min(F) + F_0)/2$ ∧ labelbin($F_1$) = true then
6: $F_0$ ← $F_1$
7: else
8: post $F > (\min(F) + F_0)/2$
9: end if
10: end while
11: return true

Algorithm 2: Optimization with binary search

PROCEDURE labelbinora($O_0$, $F$, $O$) : (false, true)
Ensure: $F$ and $O$ are output parameters.
1: if feasible($O_0$) = true then
2: $F$ ← the current cost value
3: $O$ ← the current oracle
4: display data about the solution found
5: return true
6: else
7: return false
8: end if

PROCEDURE optbinora($F$) : (false, true)
Ensure: $F$ is an output parameter, receiving the optimal cost value on success.
1: if labelbinora([], $F_0$, $O_0$) = false then
2: return false
3: end if
4: while min($F$) < $F_0$ do
5: if post $F \leq (\min(F) + F_0)/2$ ∧ labelbinora($O_0$, $F_1$, $O_1$) = true then
6: $F_0$ ← $F_1$
7: $O_0$ ← $O_1$
8: else
9: post $F > (\min(F) + F_0)/2$
10: end if
11: end while
12: return true

Algorithm 3: Optimization with binary search and oracles
We now present the test results.

*Plot no oracle.* In this plot, the mutated version does not use oracles in the search. All problem instances were speeded up by using oracles, and the greatest speed-ups were observed for the more time-consuming instances.

*Plot no binary search.* In this plot, the mutated version uses an iterated search for feasible solutions with successively tighter and tighter cost bounds. The data shows that this approach is consistently better than the binary search version, although some real-life instances are counter-examples.

*Plot one-step look-ahead.* In this plot, the mutated version uses a value-choice heuristic that ignores the $R_2$ term. The $R_2$ term has no apparent effect on the test set, but was noted to sometimes make a difference on real-life instances supplied to us by Pyrosequencing AB.

*Plot no LRU.* In this plot, the mutated version has no LRU rule for breaking ties. It shows the same lack of effect as the previous plot, but the resulting MDOs on real-life instances looked better to human experts.

![Image](image.png)

**Fig. 4.** Time in seconds to reach optimality for 184 random instances. Each plot compares the baseline version with a mutated one.
6 Conclusion

We have introduced the multiplex dispensation order generation problem, a real-life combinatorial problem that arises in the context of analyzing large numbers of short to medium length DNA sequences. We have modeled the MDO generation problem as a COP. The model has been implemented using the CLP(FD) library of SICStus Prolog [5], and we comment on the encoding of some of the constraints of the COP model. A crucial part of the implementation is a custom search procedure, which is described in detail. Finally, we provide some experimental data supporting the design choices of the search procedure.

One of the lessons learnt from this study is that different designs of the COP model ($D^n_k$ vs. $K_c$ variables) can differ a lot in the ease with which the relevant constraints are expressed. This was a crucial factor in the choice of the model that uses $D^n_k$ variables, although these have much larger domains and so would suggest a larger search space in theory.

Another lesson learnt, or reinforced, is that “the devil is in the details” when it comes to search procedures—a combination of techniques was necessary in order to achieve good performance.

Acknowledgments

The research reported herein was funded by Pyrosequencing AB. The graphics used in Fig. 1 is courtesy of the Division of Molecular Biotechnology, Royal Institute of Technology, Stockholm. We are grateful to Per Mildner and the anonymous referees for their useful comments.

References

1. Nyrén, Pettersson, and Uhlén. Solid phase DNA minisequencing by an enzymatic lumino-
2. Ronaghi, Karamohamed, Pettersson, Uhlén, and Nyrén. Real-time DNA sequencing using
In Yi-Ping Phoebe Chen, editor, *Proc. APBC2004*, volume 29 of *Conferences in Research
Society.
6. Pascal Van Hentenryck and Thierry Le Provost. Incremental Search in Constraint Logic Pro-

3 www.pyrosequencing.com
4 www.biotech.kth.se/molbio