New algorithms for DNA sequencing by hybridization*

Christian Blum† Mateu Yábar Vallès Maria J. Blesa

ALBCOM research group‡
Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Jordi Girona 1–3, Ω building, Campus Nord, E-08034 Barcelona (Spain)
{cblum,mjblesa}@lsi.upc.edu, mateuyabar@gmail.com

Abstract

The reconstruction of DNA sequences from DNA fragments is one of the most challenging problems in computational biology. In recent years the specific problem of DNA sequencing by hybridization has attracted quite a lot of interest in the optimization community. Despite the fact that well-working constructive heuristics are often the basis for well-working metaheuristics, only two constructive heuristics exist. Both approaches were proposed by Błażewicz and colleagues; the first one is a look-ahead greedy technique that was proposed in [3], and the second one—based on constructing reliable sub-sequences—was proposed in [2]. Our motivation was twofold. First, we wanted to develop better constructive heuristics. Second, on the basis of these heuristics we wanted to develop new state-of-the-art metaheuristics for DNA sequencing by hybridization.

In the first part of the paper we present our constructive heuristics. We show that the results of the best constructive heuristic are comparable to the results of existing metaheuristics, while using less computational time. In the second part of the paper we propose an ant colony optimization (ACO) approach and apply it in a so-called multi-level framework. Both, the ACO algorithm and the multi-level framework are based on our constructive heuristics. The computational results show that our algorithm is currently a state-of-the-art algorithm for DNA sequencing by hybridization.

1 Introduction

Deoxyribonucleic acid (DNA) is a molecule that contains the genetic instructions for the biological development of all cellular forms of life. Each DNA molecule consists of two (complementary) sequences of four different nucleotide bases, namely adenine (A), cytosine (C), guanine (G), and thymine (T). In mathematical terms each of these sequences can be represented as a word from the alphabet \{A,C,G,T\}. One of the most important problems in computational biology consists in determining the exact structure of a DNA molecule, called

---

*Christian Blum acknowledges support from the Spanish Ministry of Education and Science under the project CICYT TIN-2005-08818-C04-01 (OPLINK), and from the Ramón y Cajal program of the Spanish Ministry of Science and Technology of which he is a post-doctoral research fellow. Maria Blesa acknowledges support from the Spanish Ministry of Education and Science under the project CICYT TIN-2005-09198 (ASCE).
†Corresponding author
DNA sequencing. This is not an easy task, because the two DNA sequences of a DNA molecule are usually so large that they cannot be read in one piece. In 1977, 24 years after the discovery of DNA, two separate methods for DNA sequencing were developed: the chain termination method and the chemical degradation method. Later, in the late 1980’s, an alternative and much faster method called DNA sequencing by hybridization was developed (see [1, 19, 15]).

DNA sequencing by hybridization works roughly as follows. The first phase of the method consists of a chemical experiment which requires a so-called DNA array. A DNA array is a two-dimensional grid whose cells typically contain all possible DNA strands—called probes—of equal length \( l \). For example, consider a DNA array of all possible probes of length \( l = 3 \) (see also [18]):

\[
\begin{array}{cccccccccccc}
\text{AAA} & \text{AAC} & \text{AAG} & \text{AAT} & \text{CAA} & \text{CAC} & \text{CAG} & \text{CAT} \\
\text{ACA} & \text{ACC} & \text{ACG} & \text{ACT} & \text{CCA} & \text{CCC} & \text{CCG} & \text{CCT} \\
\text{AGA} & \text{AGC} & \text{AGG} & \text{AGT} & \text{CGA} & \text{CGC} & \text{CGG} & \text{CGT} \\
\text{ATA} & \text{ATC} & \text{ATG} & \text{ATT} & \text{CTA} & \text{CTC} & \text{CTG} & \text{CTT} \\
\text{GAA} & \text{GAC} & \text{GAG} & \text{GAT} & \text{TAA} & \text{TAC} & \text{TAG} & \text{TAT} \\
\text{GCA} & \text{GCC} & \text{GCG} & \text{GCT} & \text{TCA} & \text{TCC} & \text{TCG} & \text{TCT} \\
\text{GGA} & \text{GGC} & \text{GGG} & \text{GGT} & \text{TGA} & \text{TGC} & \text{TGG} & \text{TGT} \\
\text{GTA} & \text{GTC} & \text{GTG} & \text{GTT} & \text{TTA} & \text{TTC} & \text{TTG} & \text{TTT} \\
\end{array}
\]

After the generation of the DNA array, the chemical experiment is started. It consists of bringing together the DNA array with many copies of the DNA sequence to be read, also called the target sequence. Hereby, the target sequence might react with a probe on the DNA array if and only if the probe is a subsequence of the target sequence. Such a reaction is called hybridization. After the experiment the DNA array allows the identification of the probes that reacted with target sequences. This subset of probes is called the spectrum. Two types of errors may occur during the hybridization experiment:

1. **Negative errors:** Some probes do not appear in the spectrum even though they are part of the target sequence. A particular type of negative error is caused by the multiple existence of a probe in the target sequence. This cannot be detected by the hybridization experiment. Such a probe will appear at most once in the spectrum.

2. **Positive errors:** A probe of the spectrum that does not appear in the target sequence is called a positive error.

Given the spectrum, the second phase of DNA sequencing by hybridization consists of the reconstruction of the target sequence from the spectrum. Let us, for a moment, assume that the obtained spectrum is perfect, that is, free of errors. In this case, the original sequence can be reconstructed in polynomial time with an algorithm proposed by Pevzner in [20]. However, as the generated spectra generally contain negative as well as positive errors, the perfect reconstruction of the target sequence is \( NP \)-hard.

### 1.1 DNA sequencing by hybridization

The computational task of DNA sequencing by hybridization is generally expressed by optimization problems of which the optimal solutions can be shown to have a high probability to resemble the target sequence. In this work we consider the one proposed by Błażewicz et
Henceforth, let the target sequence be denoted by \( s \). The number of nucleotide bases of \( s \) shall be denoted by \( n \) (i.e., \( s \in \{A,C,G,T\}^n \)). Furthermore, the spectrum—as obtained by the hybridization experiment—is denoted by \( S = \{1, \ldots, m\} \). Remember that each \( i \in S \) is an oligonucleotide (i.e., a short DNA strand) of length \( l \) (i.e., \( i \in \{A,C,G,T\}^l \)). In general, the length of any oligonucleotide \( i \) is denoted by \( l(i) \). Let \( G = (V,A) \) be the completely connected directed graph defined by \( V = S \) (see also [18]). To each link \( a_{ij} \in A \) is assigned a weight \( o_{ij} \), which is defined as the length of the longest DNA strand that is a suffix of \( i \) and a prefix of \( j \) (i.e., the overlap). A directed Hamiltonian path \( p = (i_1, \ldots, i_k) \) in \( G \) is a directed path without loops. The length of such a path \( p \), denoted by \( l(p) \), is defined as the number of vertices (i.e., oligonucleotides) on the path. In the following we denote by \( p[r] \) the \( r \)-th vertex in a given path \( p \) (starting from position 1). In contrast to the length, the cost of a path \( p \) is defined as follows:

\[
c(p) \leftarrow l(p) \cdot l - \sum_{r=1}^{l(p)-1} o_{p[r] \ p[r+1]} \tag{1}
\]

The first term sums up the length of the oligonucleotides on the path, and the second term (which is subtracted from the first one) sums up the overlaps between the neighboring oligonucleotides on \( p \). In fact, \( c(p) \) is equivalent to the length of the DNA sequence that is obtained by the sequence of oligonucleotides in \( p \). The problem of DNA sequencing by hybridization consists of finding a directed Hamiltonian path \( p^* \) in \( G \) with \( l(p^*) \geq l(p) \) for all possible paths \( p \) that fulfill \( c(p) \leq n \). In the following we refer to this optimization problem as \textit{sequencing by hybridization (SBH)}, and we will denote an SBH problem instance by \((G,n)\).

### 1.2 Example

As an example consider the target sequence \( s_t = \text{ACTGACTC} \). Assuming \( l = 3 \), the ideal spectrum is \{ACT,CTG,TGA,GAC, ACT,CTC\}. However, let us assume that the hybridization experiment provides us with the following faulty spectrum \( S = \{\text{ACT,TGA,GAC,CTC,TAA}\} \). This spectrum has two negative errors, because ACT should appear twice, but can—due to the characteristics of the hybridization experiment—only appear once, and CTG does not appear at all in \( S \). Moreover, \( S \) has one positive error, because it includes oligonucleotide TAA, which does not appear in the target sequence. An optimal Hamiltonian path in this example is \( p^* = (\text{ACT,TGA,GAC,CTC}) \) with \( l(p^*) = 4 \) and \( c(p^*) = 8 \). The DNA sequence that is retrieved from this path is equal to the target sequence (see Figure 1).

### 1.3 Existing approaches

The first approach to solve the SBH problem was a branch & bound method proposed in [3]. However, this approach becomes unpractical with growing problem size. For example, this algorithm was only able to solve 1 out of 40 different problem instances concerning target sequences of length 209 within one hour. Another argument against this branch & bound algorithm is the fact that an optimal solution to the SBH problem does not necessarily provide a DNA sequence that is equal to the target sequence. This implies that the importance of

---

1 In [7] it was shown that this model is \(\overline{NP}\)-hard.
(a) Completely connected directed graph $G$.

(b) DNA sequence retrieval from a Hamiltonian path.

Figure 1: Example. (a) The completely connected directed graph $G$ with spectrum $S = \{ACT, TGA, GAC, CTC, TAA\}$ as the vertex set. The edge weights (i.e., overlaps) are not indicated for readability reasons. For example, the weight on the edge from TGA to GAC is 2, because GA is the longest DNA strand that is a suffix of TGA and a prefix of GAC. An optimal Hamiltonian path is $p^* = (ACT, TGA, GAC, CTC)$. In (b) is shown how to retrieve the DNA sequence that is encoded by $p^*$. Note that $c(p^*) = 8$, which is equal to the length of the encoded DNA sequence.

Finding optimal solutions is not the same as for other optimization problems. Therefore, the research community has focused on heuristic techniques for tackling the SBH problem. Most of the existing approaches are metaheuristics such as evolutionary algorithms and tabu search techniques (for a general introduction to metaheuristic methods see, for example, [10]). For an overview on the existing approaches for the SBH problem see Table 1.²

1.4 Benchmark instances

The literature on DNA sequencing by hybridization offers several benchmark instance sets. By far the most popular one was introduced by Blazewicz et al. in [3]. This instance set consists of 40 real DNA target sequences of length 109, 209, 309, 409, and 509 (altogether 200 instances). Based on real hybridization experiments, the spectra were generated with probe size $l = 10$. All spectra contain 20% negative errors as well as 20% positive errors. For example, the spectra concerning the target sequences of length 109 contain 100 oligonucleotides of which 20 oligonucleotides do not appear in the target sequences. Let this instance set in the following be denoted by Set1.

A second set of benchmark instances was introduced by Fernandes and Ribeiro in [17]. It consists of randomly generated DNA target sequences of different sizes and different oligonucleotide lengths. More in detail, $n \in \{100, 200, \ldots, 1000\}$ and $l \in \{7, 8, 9, 10\}$. For each combination of $n$ and $l$, this instance set consists of 100 target sequences; 4000 in total. All spectra contain 20% negative errors as well as 20% positive errors. Henceforth we denote this benchmark set by Set2.

²Note that the GRASP method proposed in [17] deals with an easier version of the problem in which the first oligonucleotide of each target sequence is known.
Table 1: A list of approaches for the SBH problem.

<table>
<thead>
<tr>
<th>Type of algorithm</th>
<th>Identifier</th>
<th>Publication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructive heuristic</td>
<td>LAG</td>
<td>Błażewicz et al. [3], 1999</td>
</tr>
<tr>
<td>Constructive heuristic</td>
<td>OW</td>
<td>Błażewicz et al. [2], 2002</td>
</tr>
<tr>
<td>Evolutionary algorithm</td>
<td>EA1</td>
<td>Błażewicz et al. [8, 6], 2002</td>
</tr>
<tr>
<td>Evolutionary algorithm</td>
<td>EA2</td>
<td>Endo [16], 2004</td>
</tr>
<tr>
<td>Evolutionary algorithm</td>
<td>EA3</td>
<td>Brizuela et al. [12], 2004</td>
</tr>
<tr>
<td>Evolutionary algorithm</td>
<td>EA4</td>
<td>Bui and Youssef [13], 2004</td>
</tr>
<tr>
<td>Tabu search</td>
<td>TS</td>
<td>Błażewicz et al. [4], 2000</td>
</tr>
<tr>
<td>Tabu search / scatter search hybrid</td>
<td>TS/SS</td>
<td>Błażewicz et al. [5, 6], 2004</td>
</tr>
<tr>
<td>GRASP-like multi-start technique</td>
<td>GRASP</td>
<td>Fernandes and Ribeiro [17], 2005</td>
</tr>
</tbody>
</table>

1.5 Organization of the paper

The organization of the paper is as follows. In Section 2 we present constructive heuristics, whereas in Section 3 we present an ant colony optimization approach. Furthermore, in Section 4 we propose a multi-level framework in which the ant colony optimization algorithm can be applied. Finally, in Section 5 we conduct an experimental evaluation of our algorithms and compare them to most of the techniques from the literature. In Section 6 we offer conclusions and an outlook to the future.

2 New constructive heuristics

In this section we first deal with a simple greedy technique from the literature (see [3]). Then, we propose a sensible extension of this heuristic. Finally, we present a conceptionally new heuristic that is based on merging sub-sequences. Note that a preliminar version of this section is published in [11].

Before we start the description of the heuristics we introduce some notation. In particular we use

\[
\text{pre}(i) := \arg\max\{ o_{ji} \mid j \in \hat{S}, j \neq i \}, \tag{2}
\]

\[
\text{suc}(i) := \arg\max\{ o_{ij} \mid j \in \hat{S}, j \neq i \}, \tag{3}
\]

where \( \hat{S} \subseteq S \) and \( i \in \hat{S} \) are given. In words, pre\((i)\) is the best available predecessor for \( i \) in \( \hat{S} \), that is, the oligonucleotide that—as a predecessor of \( i \)—has the biggest overlap with \( i \). Accordingly, suc\((i)\) is the best available successor for \( i \) in \( \hat{S} \). In case of ties, the first one that is found is taken.

2.1 A simple greedy technique

Given a problem instance \((G, n)\), the simple greedy technique (henceforth denoted by GREEDY) works as shown in Algorithm 1. The construction of a path \( p \) in graph \( G \) starts by choosing one of the oligonucleotides from \( S \) in function \text{Choose\_Initial\_Oligonucleotide}(S). In subsequent construction steps \( p \) is extended by adding exactly one oligonucleotide chosen in function \text{Choose\_Next}(\hat{S}). Finally, the solution construction stops as soon as \( c(p) \geq n \), that is, when
Algorithm 1 The GREEDY heuristic

1: input: A problem instance \((G, n)\)
2: \(i^* := \text{Choose Initial Oligonucleotide}(S)\)
3: \(p := (i^*)\)
4: \(\hat{S} := S\)
5: while \(c(p) < n\) do
6: \(\hat{S} := \hat{S} \setminus \{i^*\}\)
7: \(i^* := \text{Choose Next}(\hat{S})\)
8: Extend path \(p\) by adding \(i^*\) to its end
9: end while
10: \(p \leftarrow \text{Find Best Subpath}(p)\)
11: output: DNA sequence \(s\) that is obtained from \(p\)

the DNA sequence derived from the constructed path \(p\) is at least as long as the target sequence \(s_t\). In case \(c(p) > n\), function \(\text{Find Best Subpath}(p)\) searches for the longest (in terms of the number of oligonucleotides) subpath \(p'\) of \(p\) such that \(c(p') \leq n\), and replaces \(p\) by \(p'\).

Different versions of GREEDY are obtained by different implementations of the functions \(\text{Choose Initial Oligonucleotide}(S)\) and \(\text{Choose Next}(\hat{S})\). The first and only version that was published in the literature so far (see [3]) implements these functions as follows. The function \(\text{Choose Initial Oligonucleotide}(S)\) chooses an oligonucleotide uniformly at random from \(S\). Moreover, function \(\text{Choose Next}(\hat{S})\) chooses the oligonucleotide \(i^*\) such that

\[
i^* := \arg\max \{o_{p[l(p)]i} + o_{i\text{succ}(i)} \mid i \in \hat{S}\}, \tag{4}\]

where \(p\) is the current path. Note that this implementation realizes a look-ahead strategy.

In this work we will study the behaviour of two versions of GREEDY. Both versions use the following implementation of function \(\text{Choose Initial Oligonucleotide}(S)\). First, \(S_{bs} \subseteq S\) is defined as the set of all oligonucleotides in \(S\) whose best successor is better or equal to the best successor of the all the other oligonucleotides in \(S\), that is

\[
S_{bs} := \{i \in S \mid o_{i\text{succ}(i)} \geq o_{j\text{succ}(j)}, j \in S\}. \tag{5}\]

Second, \(S_{wp} \subseteq S_{bs}\) is defined as the set of all oligonucleotides in \(S_{bs}\) whose best predecessor is worse or equal to the best predecessor of all the other oligonucleotides in \(S_{bs}\), that is

\[
S_{wp} := \{i \in S_{bs} \mid o_{i\text{pred}(i)} \leq o_{j\text{pred}(j)}, j \in S_{bs}\}. \tag{6}\]

As starting oligonucleotide we choose the one (from \(S_{wp}\)) that is found first. The idea hereby is to start the path construction with an oligonucleotide that has a very good successor and at the same time a very bad predecessor. Such an oligonucleotide has a high probability to coincide with the start of the target sequence \(s_t\).

Our two versions of GREEDY differ in the implementation of function \(\text{Choose Next}(\hat{S})\). While the simple version (henceforth denoted by \(\text{GREEDY}(S)\)) chooses the oligonucleotide \(i^*\) such that \(i^* := \text{suc}(pl[p])\), the look-ahead version (henceforth denoted by \(\text{GREEDY}(LAG)\)) chooses \(i^*\) as shown in Equation 4.
2.2 An extended Greedy heuristic

A simple extension of the GREEDY heuristic outlined in the previous section is obtained by allowing the path construction not only in forward direction but also in backward direction. We denote this heuristic henceforth by FB-GREEDY. At each construction step the heuristic decides to extend the current path either in forward direction or in backward direction (see Algorithm 2). A second change with respect to GREEDY concerns the implementation of function Choose Initial Oligonucleotide(S). As the path construction allows forward and backward construction it is not necessary to start the path construction with an oligonucleotide that has a high probability of being the beginning of the target sequence. It is more important to start with an oligonucleotide that has a high probability of being part of the target sequence. We will study the following two different versions of FB-GREEDY.

The simple version (henceforth denoted by FB-GREEDY(S)) implements as follows the function Choose Initial Oligonucleotide(S):

\[ i^* := \arg \max_{i \in S} \{ o_{\text{pre}(i)} + o_{\text{suc}(i)} \mid i \in S \} \]  
(7)

In case of ties, the first one found is taken. Function Choose Forward(\(\hat{S}\)) chooses \(i^{fw}\) such that \(i^{fw} := \text{suc}(p[l(p)])\), whereas function Choose Backward(\(\hat{S}\)) chooses \(i^{bw}\) such that \(i^{bw} := \text{pre}(p[1])\). In line 8 of Algorithm 2, \(i^{fw}\) is better than \(i^{bw}\) if \(o_{\text{suc}(i^{fw})} > o_{\text{pre}(i^{bw})}\).

In contrast, the look-ahead version of FB-GREEDY (henceforth denoted by FB-GREEDY(LAG)) implements function Choose Initial Oligonucleotide(S) as follows:

\[ i^* := \arg \max_{i \in S} \{ o_{\text{pre}^2(i)} + o_{\text{pre}(i)} + o_{\text{suc}(i)} + o_{\text{suc}^2(i)} \} \]  
(8)

where \(\text{pre}^2(i)\) denotes the best predecessor of the best predecessor of \(i\) (i.e., \(\text{pre}(\text{pre}(i))\)), and similar for \(\text{suc}^2(i)\). Moreover, function Choose Forward(\(\hat{S}\)) chooses \(i^{fw}\) according to Equation 4, whereas function Choose Backward(\(\hat{S}\)) selects \(i^{bw}\) via a corresponding look-ahead strategy in backward direction:

\[ i^{bw} := \arg \max_{i \in \hat{S}} \{ o_{\text{pre}(i)} + o_{\text{p}[1]} \mid i \in \hat{S} \} \]  
(9)

Finally, in line 8 of Algorithm 2, \(i^{fw}\) is better than \(i^{bw}\) if, and only if,

\[ o_{p[l(p)]} i^{fw} + o_{i^{fw} \text{suc}(i^{fw})} > o_{\text{pre}(i^{bw})} i^{bw} + o_{i^{bw} \text{p}[1]} \]

2.3 The sub-sequence merger heuristic

The idea of the sub-sequence merger (SM) heuristic (see Algorithm 3) is conceptionally quite different to the greedy heuristics presented before. Instead of constructing only one path, the heuristic starts with a set of \(|S|\) paths, each of which only contains exactly one oligonucleotide \(i \in S\). In subsequent steps the heuristic merges paths until a path of sufficient size is obtained. The heuristic works in two phases. In the first phase, two paths \(p\) and \(p'\) can only be merged if \(p'\) is the unique best successor of \(p\), and if \(p\) is the unique best predecessor of \(p'\). The heuristic enters into the second phase if and only if the first phase has not already produced a path.
Algorithm 2 The FB-GREEDY heuristic

1: **input:** A problem instance \((G, n)\)
2: \(i^* := \text{Choose Initial Oligonucleotide}(S)\)
3: \(p := (i^*)\)
4: \(\hat{S} := \hat{S} \setminus \{i^*\}\)
5: **while** \(c(p) < n\) **do**
6: \(i_{fw} := \text{Choose Forward}(\hat{S})\)
7: \(i_{bw} := \text{Choose Backward}(\hat{S})\)
8: **if** \(i_{fw}\) better than \(i_{bw}\) **then**
9: Extend path \(p\) by adding \(i_{fw}\) to its end
10: \(\hat{S} := \hat{S} \setminus \{i_{fw}\}\)
11: **else**
12: Extend path \(p\) by adding \(i_{bw}\) to its beginning
13: \(\hat{S} := \hat{S} \setminus \{i_{bw}\}\)
14: **end if**
15: **end while**
16: \(p := \text{Find Best Subpath}(p)\)
17: **output:** DNA sequence \(s\) that is obtained from \(p\)

of sufficient length. In the second phase, the uniqueness conditions are relaxed, that is, two paths \(p\) and \(p'\) can be merged if \(p'\) is among the best successors of \(p\), and \(p\) is among the best predecessors of \(p'\). The reason of having two phases is the following: The first phase aims to produce possibly error free sub-sequences of the target sequence, whereas the second phase (which is more error prone due to the relaxed uniqueness condition) aims at connecting the sub-sequences produced in the first phase in a reasonable way.

In Algorithm 3, given two paths \(p\) and \(p'\), \(o_{pp'}\) is defined as \(o_{p \mid p,p' \in P, p' \neq p}\), that is, the overlap of the last oligonucleotide in \(p\) with the first one in \(p'\). In correspondence to the notations introduced in Equations 2 and 3, the following notations are used:

\[
\begin{align*}
\text{suc}(p) & := \arg\max\{o_{pp'} \mid p' \in P, p' \neq p\}, \\
\text{pre}(p) & := \arg\max\{o_{p'p} \mid p' \in P, p' \neq p\}.
\end{align*}
\]

Furthermore, \(S_{\text{suc}}(p)\) is defined as the set of best successors of \(p\), that is,

\[
S_{\text{suc}}(p) := \{p' \in P \mid o_{p'p} = o_{p\text{suc}(p)}\};
\]

and \(S_{\text{pre}}(p)\) is defined as the set of best predecessors of \(p\), that is,

\[
S_{\text{pre}}(p) := \{p' \in P \mid o_{p'p} = o_{\text{pre}(p)p}\}.
\]

Finally, function \(\text{Find Best Subpath}(p)\) is implemented to retrieve from path \(p\) the longest sub-path (in terms of the number of oligonucleotides).

### 2.4 Hybridizing SM with FB-GREEDY

The SM heuristic can be hybridized with the heuristics from Sections 2.1 or 2.2, based on the following observation: At every stage of the SM heuristic, a greedy heuristic can be applied
Algorithm 3 The SM heuristic

1: **input:** A problem instance \((G, n)\)

2: \(P := \{(i) \mid i \in S\}\)

3: **PHASE 1:**

4: \(stop = \text{FALSE}\)

5: **for** \(overlap = l - 1, \ldots , 1\) **do**

6: \(\text{while } \exists p, p' \in P \text{ s.t. } o_{p,p'} = overlap \& |S_{\text{suc}}(p)| = 1 \& |S_{\text{pre}}(p')| = 1 \& \text{suc}(p) = p' \& \text{pre}(p') = p \& stop = \text{FALSE} \text{ do} \)

7: \quad Add path \(p'\) to the end of path \(p\)

8: \quad \(P := P \setminus \{p'\}\)

9: \quad \text{if } c(p) \geq n \text{ then}

10: \quad \quad \text{stop} = \text{TRUE}

11: \quad \text{end if}

12: \text{end while}

13: **end for**

14: **PHASE 2:**

15: **for** \(overlap = l - 1, \ldots , 1\) **do**

16: \(\text{while } \exists p, p' \in P \text{ s.t. } o_{p,p'} = overlap \& p' \in S_{\text{suc}}(p) \& p \in S_{\text{pre}}(p') \& stop = \text{FALSE} \text{ do} \)

17: \quad Choose \(p\) and \(p'\) such that \(l(p) + l(p')\) is maximal

18: \quad Add path \(p'\) to the end of path \(p\)

19: \quad \text{if } c(p) \geq n \text{ then}

20: \quad \quad \text{stop} = \text{TRUE}

21: \quad \text{end if}

22: \text{end while}

23: **end for**

24: Let \(p\) be the path in \(P\) with maximal cost

25: \(p := \text{Find}_\text{Best}_\text{Subpath}(p)\)

26: **output:** DNA sequence \(s\) that is obtained from \(p\)

---

Note that this does not change the overlap values, that is, the overlap between two oligonucleotides \(i, j \in \hat{S}\) is the overlap between the last original oligonucleotide of \(i\) and the first original oligonucleotide from \(j\).
the performance of 2 different hybrid algorithms in this work: (1) The hybridization with FB-GREEDY(S), and (2) with FB-GREEDY(LAG).

2.5 Results of the constructive heuristics

We implemented our constructive heuristics in ANSI C++ using GCC 3.2.2 for compiling the software. Our experimental results were obtained on a PC with an AMD64X2 4400 processor and 4 Gb of memory. Then we applied these constructive heuristics to all problem instances of benchmark set Set1 (see Section 1.4). The results are shown numerically in Table 2, and graphically in Figure 2. Each sub-table contains the results of one of the heuristics. In each column of each sub-table the results are presented as averages over the 40 problem instances of the respective target sequence length. The second row of each sub-table contains the average solution quality (i.e., the average number of oligonucleotides in the constructed paths). Remember that the optimization objective of the SBH problem is to maximize this value. The third table row provides the number (out of 40) of solved problem instances, that is, the number of instances for which a path of maximal length could be found.\footnote{We would like to point out to the reader that an optimal solution to the SBH problem does not necessarily correspond to a DNA sequence that is equal to the target sequence.} The fourth and fifth table row provide average similarity scores obtained by comparing the computed DNA sequences with the target sequences. The average scores in the fourth table row are obtained by the Needleman-Wunsch algorithm, which is an algorithm for global alignment. In contrast, the average scores that are displayed in the fifth table row are obtained by the application of the Smith-Waterman algorithm, which is an algorithm for local alignment. The local alignment scores are given for completeness. Both algorithms were applied with the following parameters: +1 for a match of nucleotide bases, -1 for a mismatch or a gap. Finally, the sixth table row provides the average computation times for solving one instance (in seconds).

From the results displayed in Table 2 we can draw the following conclusions:

1. Surprisingly, the results of GREEDY(S), resp. FB-GREEDY(S), are in general not worse than the results of GREEDY(LAG), resp. FB-GREEDY(LAG). In particular, GREEDY(S) seems to have slight advantages over GREEDY(LAG) in terms of similarity scores, and slight disadvantages in terms of solutions quality. For what concerns the FB-* versions of these two heuristics, FB-GREEDY(S) seems to have advantages over FB-GREEDY(LAG) in terms of the number of instances solved to optimality. However, in terms of similarity scores and solution quality FB-GREEDY(LAG) has slight advantages over FB-GREEDY(S). This leads us to the conclusion that the look-ahead strategy is not necessary for this type of constructive heuristic. This is good news, because the simple versions only spend about half of the computation time.

2. The results of the FB-* heuristics improve in general over the results of their one-directional counterparts. This means that it is beneficial to allow the path construction in two directions (forward as well as backward).

3. The results of SM are clearly better than the results of all heuristics that construct solutions sequentially. Also in terms of computation time, SM is not much more expensive than the sequential construction heuristics.
Table 2: Results of the constructive heuristics for benchmark instances of Set1, that is, the instances by Błażewicz et al. [3].

<table>
<thead>
<tr>
<th>Spectrum size</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average solution quality</td>
<td>77.20</td>
<td>152.78</td>
<td>229.00</td>
<td>302.73</td>
<td>375.28</td>
</tr>
<tr>
<td>Solved instances</td>
<td>26</td>
<td>18</td>
<td>18</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Average similarity score (global)</td>
<td>80.85</td>
<td>142.10</td>
<td>199.98</td>
<td>214.05</td>
<td>269.68</td>
</tr>
<tr>
<td>Average similarity score (local)</td>
<td>95.28</td>
<td>167.20</td>
<td>234.80</td>
<td>277.33</td>
<td>334.90</td>
</tr>
<tr>
<td>Average computation time (sec)</td>
<td>0.0028</td>
<td>0.012</td>
<td>0.025</td>
<td>0.044</td>
<td>0.069</td>
</tr>
</tbody>
</table>

(a) Results of GREEDY(S)

<table>
<thead>
<tr>
<th>Spectrum size</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average solution quality</td>
<td>76.98</td>
<td>153.53</td>
<td>230.68</td>
<td>309.03</td>
<td>383.08</td>
</tr>
<tr>
<td>Solved instances</td>
<td>23</td>
<td>15</td>
<td>12</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>Average similarity score (global)</td>
<td>77.05</td>
<td>133.63</td>
<td>171.78</td>
<td>206.80</td>
<td>218.60</td>
</tr>
<tr>
<td>Average similarity score (local)</td>
<td>91.83</td>
<td>152.43</td>
<td>209.33</td>
<td>272.40</td>
<td>293.48</td>
</tr>
<tr>
<td>Average computation time (sec)</td>
<td>0.0035</td>
<td>0.016</td>
<td>0.037</td>
<td>0.076</td>
<td>0.13</td>
</tr>
</tbody>
</table>

(b) Results of GREEDY(LAG)

<table>
<thead>
<tr>
<th>Spectrum size</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average solution quality</td>
<td>77.65</td>
<td>148.28</td>
<td>218.33</td>
<td>299.30</td>
<td>355.65</td>
</tr>
<tr>
<td>Solved instances</td>
<td>36</td>
<td>21</td>
<td>19</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>Average similarity score (global)</td>
<td>102.28</td>
<td>159.48</td>
<td>213.50</td>
<td>247.20</td>
<td>226.03</td>
</tr>
<tr>
<td>Average similarity score (local)</td>
<td>104.13</td>
<td>179.48</td>
<td>252.10</td>
<td>303.68</td>
<td>319.40</td>
</tr>
<tr>
<td>Average computation time (sec)</td>
<td>0.004</td>
<td>0.014</td>
<td>0.032</td>
<td>0.059</td>
<td>0.091</td>
</tr>
</tbody>
</table>

(c) Results of FB-GREEDY(S)

<table>
<thead>
<tr>
<th>Spectrum size</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average solution quality</td>
<td>78.38</td>
<td>155.70</td>
<td>234.95</td>
<td>310.03</td>
<td>386.20</td>
</tr>
<tr>
<td>Solved instances</td>
<td>32</td>
<td>17</td>
<td>18</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Average similarity score (global)</td>
<td>99.78</td>
<td>153.03</td>
<td>225.45</td>
<td>241.00</td>
<td>221.83</td>
</tr>
<tr>
<td>Average similarity score (local)</td>
<td>102.38</td>
<td>174.15</td>
<td>253.63</td>
<td>303.68</td>
<td>316.68</td>
</tr>
<tr>
<td>Average computation time (sec)</td>
<td>0.0051</td>
<td>0.022</td>
<td>0.054</td>
<td>0.11</td>
<td>0.19</td>
</tr>
</tbody>
</table>

(d) Results of FB-GREEDY(LAG)

<table>
<thead>
<tr>
<th>Spectrum size</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average solution quality</td>
<td>79.75</td>
<td>157.80</td>
<td>234.90</td>
<td>306.90</td>
<td>367.38</td>
</tr>
<tr>
<td>Solved instances</td>
<td>38</td>
<td>31</td>
<td>30</td>
<td>28</td>
<td>18</td>
</tr>
<tr>
<td>Average similarity score (global)</td>
<td>106.33</td>
<td>195.85</td>
<td>284.68</td>
<td>357.98</td>
<td>376.25</td>
</tr>
<tr>
<td>Average similarity score (local)</td>
<td>107.20</td>
<td>203.03</td>
<td>293.75</td>
<td>377.00</td>
<td>416.68</td>
</tr>
<tr>
<td>Average computation time (sec)</td>
<td>0.005</td>
<td>0.004</td>
<td>0.046</td>
<td>0.082</td>
<td>0.13</td>
</tr>
</tbody>
</table>

(e) Results of SM

<table>
<thead>
<tr>
<th>Spectrum size</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average solution quality</td>
<td>80.00</td>
<td>159.55</td>
<td>238.68</td>
<td>318.40</td>
<td>394.55</td>
</tr>
<tr>
<td>Solved instances</td>
<td>40</td>
<td>36</td>
<td>36</td>
<td>32</td>
<td>28</td>
</tr>
<tr>
<td>Average similarity score (global)</td>
<td>108.40</td>
<td>204.55</td>
<td>285.63</td>
<td>375.88</td>
<td>425.03</td>
</tr>
<tr>
<td>Average similarity score (local)</td>
<td>108.70</td>
<td>206.85</td>
<td>298.73</td>
<td>387.33</td>
<td>456.58</td>
</tr>
<tr>
<td>Average computation time (sec)</td>
<td>0.0073</td>
<td>0.032</td>
<td>0.076</td>
<td>0.14</td>
<td>0.23</td>
</tr>
</tbody>
</table>

(f) Results of SM-FB-GREEDY(S)

<table>
<thead>
<tr>
<th>Spectrum size</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average solution quality</td>
<td>80.00</td>
<td>159.68</td>
<td>239.90</td>
<td>319.38</td>
<td>398.88</td>
</tr>
<tr>
<td>Solved instances</td>
<td>40</td>
<td>36</td>
<td>39</td>
<td>35</td>
<td>31</td>
</tr>
<tr>
<td>Average similarity score (global)</td>
<td>108.40</td>
<td>204.78</td>
<td>300.00</td>
<td>396.90</td>
<td>469.55</td>
</tr>
<tr>
<td>Average similarity score (local)</td>
<td>108.70</td>
<td>206.85</td>
<td>305.35</td>
<td>399.85</td>
<td>479.88</td>
</tr>
<tr>
<td>Average computation time (sec)</td>
<td>0.012</td>
<td>0.048</td>
<td>0.11</td>
<td>0.21</td>
<td>0.35</td>
</tr>
</tbody>
</table>

(g) Results of SM-FB-GREEDY(LAG)
4. The best results are obtained by the hybrid heuristics SM-FB-GREEDY(S) and SM-FB-GREEDY(LAG). While maintaining low computation times, these hybrids improve greatly over the pure heuristics. When comparing between the two hybrid methods, we notice that here the look-ahead strategy seems beneficial, that is, SM-FB-GREEDY(LAG) has in all quality measures (except for computation time) advantages over SM-FB-GREEDY(S).

Note that in Figure 2 we have added the results of the OW heuristic (see Table 1) to the comparison in order to show that SM-FB-GREEDY(LAG) is currently the best available constructive heuristic. Finally, Figure 3 presents a comparison between SM-FB-GREEDY(LAG) and most of the available metaheuristic approaches. The results are surprising: SM-FB-GREEDY(LAG) is clearly better than the five metaheuristic approaches EA1, EA4, TS, and TS/SS. Furthermore, the results of SM-FB-GREEDY(LAG) are—except for the problem instance of target sequence size 509—comparable to the results of the best existing metaheuristics EA2. Taking into account the advantage in computation time (i.e., HSM needs not even half a second to compute its results for the largest problem instances, while EA2—which is by far the fastest among the existing meta-heuristics—needs several seconds) this hybrid heuristic seems to be a good choice even when compared to metaheuristic approaches.

3 Ant colony optimization for DNA sequencing by hybridization

Based on our constructive heuristics, we propose in this section an ant colony optimization (ACO) algorithm [14] for the SBH problem. ACO algorithms are iterative stochastic search techniques which tackle an optimization problem as follows. At each iteration candidate solutions are constructed in a probabilistic way. The probabilistic solution construction is based on a so-called pheromone model (denoted by $T$), which is a set of numerical values that encode the algorithms’ search experience. After the construction phase, some of the generated solutions are used to update the pheromone values in a way that aims at biasing the future solution construction towards good solutions found during the search process.

3.1 The objective function

Before we outline our particular ACO implementation for SBH, we first deal with an issue concerning the objective function. Given a feasible solution $p$ to the problem instance $(G, n)$, the original objective function value $l(p)$ is the number of oligonucleotides in $p$. This objective function has the following disadvantage when used in a search algorithm. Let $p$ and $p'$ be two solutions with $l(p) = l(p')$ and $c(p) < c(p')$. Even though the objective function $l(\cdot)$ can not distinguish between $p$ and $p'$, the intuition is to prefer $p$, because the induced DNA sequence is shorter. This implies a higher chance for an extension of $p$ while respecting the constraint $c(p) \leq n$. Therefore, we define a comparison operator $f(\cdot)$ as follows:

$$f(p) > f(p') \iff l(p) > l(p') \text{ or } (l(p) = l(p') \text{ and } c(p) < c(p'))$$

5Remeber that $G$ is the completely connected, directed graph whose node set is the spectrum $S$, and $n$ is the length of the target sequence $s_t$. A solution $p$ is a Hamiltonian path in $G$.

6Remeber that $c(p)$ denotes the length of the DNA sequence derived from $p$. See Figure 1(b) for an example.
Figure 2: Comparison of all existing heuristics concerning (a) the average similarity score obtained, and (b) the number of optimally solved instances, for the instances by Błazewicz et al. [3] (Set1).
3.2 The algorithm

Our ACO approach, which is a \textit{MAX-MIN} ant system (MMAS) implemented in the hyper-cube framework (HCF)\cite{9}, solves the SBH problem as shown in Algorithm 4. The data structures used by this algorithm, in addition to counters and to the pheromone model \(T\), are:

- the \textit{iteration-best} solution \(p_{ib}\): the best solution generated in the current iteration by the ants;
- the \textit{best-so-far} solution \(p_{bs}\): the best solution generated since the start of the algorithm;
- the \textit{restart-best} solution \(p_{rb}\): the best solution generated since the last restart of the algorithm;
- the \textit{convergence factor} \(cf\), \(0 \leq cf \leq 1\): a measure of how far the algorithm is from convergence;
- the Boolean variable \(bs\_update\): it becomes true when the algorithm reaches convergence.

The algorithm works as follows. First, all the variables are initialized, and the pheromone values are set to their initial value 0 in procedure \texttt{InitializePheromoneValues} \((T)\). At each iteration, first \(n_f\) ants construct a solution each in procedure \texttt{ConstructForwardSolution} \((T)\), and then \(n_b\) ants construct a solution each in procedure \texttt{ConstructBackwardSolution} \((T)\). A forward solution is constructed from left to right, and a backward solution from right to left. Subsequently, the value of the variables \(p_{ib}\), \(p_{rb}\) and \(p_{bs}\) is updated (note that, until the first restart of the algorithm, it holds that \(p_{rb} \equiv p_{bs}\)). Fourth, pheromone values are updated via the \texttt{ApplyPheromoneUpdate} \((cf, bs\_update, T, p_{ib}, p_{rb}, p_{bs})\) procedure. Fifth, a new value for the convergence factor \(cf\) is computed. Depending on this value, as well as on the value of the Boolean variable \(bs\_update\), a decision on whether to restart the algorithm or not is taken. If the algorithm is restarted, the procedure \texttt{ResetPheromoneValues} \((T)\) is applied and all the pheromones are reset to their initial value \((0.5)\). The algorithm is iterated until some opportunely defined termination conditions are satisfied. Once terminated the algorithm returns the best-so-far solution \(p_{bs}\). The main procedures of Algorithm 4 are now described in detail.

\texttt{ConstructForwardSolution} \((T)\): This function constructs a path \(p = (i_1, \ldots, i_k)\) in \(G\) from left to right by using a probabilistic version of the \textsc{Greedy} heuristic (see Algorithm 1). Both functions \texttt{Choose InitialOligonucleotide} \((S)\) and \texttt{Choose Next} \((\hat{S})\) of Algorithm 1 utilize a pheromone model \(T\), which consists of pheromone values \(\tau_{ij}\) and \(\tau_{ji}\) for each pair \(i, j \in S\) \((i \neq j)\), that is, to each directed link of \(G\) is associated a pheromone value. Additionally, \(T\) comprises pheromone values \(\tau_{0i}\) and \(\tau_{i0}\) for all \(i \in S\), where 0 is a non-existing dummy oligonucleotide.

Function \texttt{Choose Next} \((\hat{S})\) is implemented as follows. First, a desirability value \(\mu_{ij} := \tau_{ij} \cdot |\eta_{ij}|^5\) is computed for all \(j \in \hat{S}\), where \(\eta_{ij} := o_{ij}/(l-1)\). The values \(\eta_{ij}\) are called \textit{heuristic information}. They are defined such that \(\eta_{ij} \in [0,1]\) grows with growing overlap \(o_{ij}\) between the oligonucleotides \(i\) and \(j\). Note that when the pheromone values are all equal, the desirability value \(\mu_{ij}\) is high exactly when \(o_{ij}\) is high. Then, we generate a so-called \textit{restricted candidate list} \(S^{\text{rl}} \subseteq \hat{S}\) with a pre-defined cardinality \(cls\) such that \(\mu_{ij} \geq \mu_{iu}\) for all...
Algorithm 4 ACO for the SBH problem

**Input:** a problem instance \((G, n)\)

\[ p_{bs} \leftarrow \text{NULL} \]

\[ p_{rb} \leftarrow \text{NULL} \]

\[ cf \leftarrow 0 \]

\[ bs\_update \leftarrow \text{FALSE} \]

**InitializePheromoneValues** \((T)\)

**While** termination conditions not satisfied **do**

**For** \( j \leftarrow 1 \) to \( n_f \) **do**

**Let** \( p_j \leftarrow \text{ConstructForwardSolution}(T) \)

**EndFor**

**For** \( j \leftarrow n_f + 1 \) to \( n_f + n_b \) **do**

**Let** \( p_j \leftarrow \text{ConstructBackwardSolution}(T) \)

**EndFor**

**Let** \( p_{ib} \leftarrow \text{argmax} \{ f(p_1), ..., f(p_{n_f+n_b}) \} \)

**If** \( p_{rb} = \text{null} \) or \( f(p_{ib}) > f(p_{rb}) \) **then**

**Let** \( p_{rb} \leftarrow p_{ib} \)

**Else**

**Let** \( p_{bs} \leftarrow p_{ib} \)

**EndIf**

**ApplyPheromoneUpdate** \((cf, bs\_update, T, p_{ib}, p_{rb}, p_{bs})\)

**Let** \( cf \leftarrow \text{ComputeConvergenceFactor}(T) \)

**If** \( cf > 0.9999 \) **then**

**If** \( bs\_update = \text{TRUE} \) **then**

**ResetPheromoneValues** \((T)\)

**Let** \( p_{rb} \leftarrow \text{null} \)

**Let** \( bs\_update \leftarrow \text{FALSE} \)

**Else**

**Let** \( bs\_update \leftarrow \text{TRUE} \)

**EndIf**

**EndIf**

**EndWhile**

**Output:** \( p_{bs} \)

\[ j \in \hat{S} \text{ and } u \in S^\text{rel}. \text{ Then, with probability } q \in [0, 1) \text{ the next oligonucleotide } i_{t+1} \text{ is chosen from } S^\text{rel} \text{ such that} \]

\[ i_{t+1} := \text{arg max}_{j \in S^\text{rel}} \{ \mu_{i_{t+1}} \} . \]  \hspace{1cm} (13)

Otherwise, the next oligonucleotide \( i_{t+1} \) is chosen from \( S^\text{rel} \) by roulette-wheel-selection according to the following probabilities:

\[ p_{i_{t+1}} := \frac{\mu_{i_{t+1}}}{\sum_{u \in S^\text{rel}} \mu_{i_{t+1}}} \]  \hspace{1cm} (14)

Note that \( q \) (henceforth called the determinism rate) and \( cls \) are important parameters of the algorithm.

In contrast to function \( \text{Choose\_Next}(\hat{S}) \), in function \( \text{Choose\_Initial\_Oligonucleotide}(S) \) the desirability values are computed as \( \mu_{i_{t+1}} := \eta_{i_{t+1}} \cdot [\eta_{i_{t+1}}]_{\hat{S}} \in [0, 1] \) for all \( j \in \hat{S} \) (note that \( \hat{S} = S \) when \( p = () \)). Hereby,

\[ \eta_{i_{t+1}} := \frac{l - o_{\text{pre}(j)} j + o_{\text{suc}(j)}}{2(l - 1)} . \]  \hspace{1cm} (15)
Note that this way of defining the heuristic information favours oligonucleotides that have a very good “best successor”, and at the same time a bad “best predecessor”. The intuition is that the spectrum most probably does not contain an oligonucleotide that is a good predecessor for the first oligonucleotide of the target sequence. Having defined the desirability values for the first construction step, the further procedure concerning the derivation of the restricted candidate list $S^{rel}$ and the choice of one of the oligonucleotides from $S^{rel}$ is the same as outlined above for function Choose_Next($\hat{S}$).

\textbf{ConstructBackwardSolution}(T): This function works principally in the same way as does function ConstructForwardSolution(T). The first difference is that a solution $p$ is constructed from right to left. The second difference is that—given a partial solution $p = (i_1, \ldots, i_l)$—the desirability values are still computed as if the solution construction were from left to right. For example, the desirability value of adding an oligonucleotide $j$ to the front of $p$ is $\mu_{ji}$ (instead of $\mu_{ij}$). This is done such that for the construction of a solution $p$ the same pheromone values are used, no matter if the solution is constructed from left to right, or from right to left.

\textbf{ApplyPheromoneUpdate}(cf, bs_update, $T$, $p_{ib}, p_{rb}, p_{bs}$): As usual for MMAS implementations in the HCF, we use at each iteration a weighted combination of the solutions $p_{ib}$, $p_{rb}$, and $p_{bs}$ for updating the pheromone values. The weight of each solution depends on the value of the convergence factor $cf$ and on the Boolean variable bs_update. In general, the pheromone update is performed as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \rho \cdot (m_{ij} - \tau_{ij}) \quad \forall \tau_{ij} \in T$$

where $\rho \in (0,1]$ is a constant called learning rate, and $m_{ij}$ is composed as follows:

$$m_{ij} \leftarrow (\kappa_{ib} \cdot \delta_{ij}(p_{ib})) + (\kappa_{rb} \cdot \delta_{ij}(p_{rb})) + (\kappa_{bs} \cdot \delta(p_{bs}))$$

where $\kappa_{ib}$ is the weight of solution $p_{ib}$, $\kappa_{rb}$ is the weight of solution $p_{rb}$, $\kappa_{bs}$ is the weight of solution $p_{bs}$, and $\kappa_{ib} + \kappa_{rb} + \kappa_{bs} = 1$. Moreover, when $i \neq 0$ and $j \neq 0$, $\delta_{ij}(p)$ is a function that returns 1 in case $j$ is the direct successor of $i$ in $p$, and 0 otherwise. In case $i = 0$, $\delta_{ij}(p)$ returns 1 in case $j$ is the first oligonucleotide in $p$, and 0 otherwise. In case $j = 0$, $\delta_{ij}(p)$ returns 1 in case $i$ is the last oligonucleotide in $p$, and 0 otherwise. After the pheromone update rule (Equation 16) is applied, pheromone values that exceed an upper limit of $\tau_{max} = 0.99$ are set back to $\tau_{max}$, and pheromone values that fall below a lower limit $\tau_{min} = 0.01$ are set back to $\tau_{min}$. This prevents the algorithm from complete convergence.

Equation 17 allows to choose how to schedule the relative influence of the three solutions used for updating pheromones. The exact schedule for the setting of the three solution weights used by MMAS in the HCF is shown in Table 3. In the early stages of the search (i.e., when $cf < 0.7$), only the iteration-best solution is used. Then, when the value of the convergence factor increases (i.e., $0.7 \leq cf < 0.9$) one third of the total influence is given to the restart-best solution, which then increases to two thirds when $0.9 \leq cf < 0.95$. Eventually, all the influence is given to the restart-best solution (i.e., when $cf \geq 0.95$). Once the value of the convergence factor raises above 0.9999, the Boolean control variable bs_update is set to TRUE, and all the influence is given to the best-so-far solution.
Table 3: Setting of $\kappa_{ib}$, $\kappa_{rb}$ and $\kappa_{bs}$ depending on the convergence factor $cf$ and the Boolean control variable $bs\_update$.

<table>
<thead>
<tr>
<th></th>
<th>$bs_update = \text{false}$</th>
<th>$bs_update = \text{true}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$cf &lt; 0.7$</td>
<td>$cf \in [0.7, 0.9]$</td>
<td>$cf \in [0.9, 0.95]$</td>
</tr>
<tr>
<td>$\kappa_{ib}$</td>
<td>1</td>
<td>2/3</td>
</tr>
<tr>
<td>$\kappa_{rb}$</td>
<td>0</td>
<td>1/3</td>
</tr>
<tr>
<td>$\kappa_{bs}$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

ComputeConvergenceFactor$(T)$: The convergence factor $cf$, which is a function of the current pheromone values, is computed as follows:

$$cf \leftarrow 2 \left( \frac{\sum_{\tau_{ij} \in T} \max\{\tau_{max} - \tau_{ij}, \tau_{ij} - \tau_{min}\}}{|T| \cdot (\tau_{max} - \tau_{min})} \right) - 0.5$$

This formula says that when the algorithm is initialized (or reset) so that all pheromone values are set to 0.5, then $cf = 0$, while when the algorithm has converged, then $cf = 1$. In all other cases, $cf$ has a value in $(0, 1)$.

4 A multi-level framework

The basic idea of a multilevel framework (see [21]) is a simple one. Starting from some given problem instance, smaller and smaller problem instances are obtained by successive coarsening until some stopping criteria are satisfied. For example, in graph-based problems the coarsening of a problem instance is usually obtained by edge contractions. This creates a hierarchy of problem instances in which the problem instance of a given level is always smaller (or equal) to the problem instance of the next higher level. Then, a solution is computed to the smallest problem instance and successively transformed into a solution of the next higher level until a solution for the original problem instance is obtained. At each level, the obtained solution might be subject to a refinement process. In our case, we will use the ACO algorithm outlined in the previous section as refinement process at each level.

4.1 Instance contraction

As already described in Section 2.4, heuristic SM provides us with a natural instance contraction mechanism for SBH. However, in order to produce possibly error-free instance contractions, we only use the first phase of SM for the multi-level framework. Each construction step leads to a new set of paths $P$ from which a new (smaller) problem instance is generated in function GenerateProblemInstance$(P)$. This is done by deriving from each path $p \in P$ the corresponding DNA strand (as exemplary shown in Figure 1(b)). This mechanism generates a sequence $(G_0, n), (G_1, n), \ldots, (G_d, n)$ of smaller and smaller problem instances (where $(G_0, n)$ is the original instance $(G, n)$). $(G_d, n)$ denotes the smallest instances that can be obtained (that is, a further construction step would produce a path $p$ with $c(p) \geq n$). Note that all these problem instances have the same target sequence. Moreover, a solution to any of these
Algorithm 5 Instance contraction

1: **input:** a problem instance \((G, n)\)
2: \(P \leftarrow \{(i) \mid i \in S\}\)
3: \(stop = \text{FALSE}\)
4: \(level = 1\)
5: for \(overlap = l - 1, \ldots, 1\) do
6: \(changed = \text{FALSE}\)
7: while \(\exists p, p' \in P \text{ s.t. } o_{pp'} = overlap \& |S_{\text{suc}}(p)| = 1 \& |S_{\text{pre}}(p')| = 1 \& \text{suc}(p) = p' \& \text{pre}(p') = p \& stop = \text{FALSE}\) do
8: \(changed = \text{TRUE}\)
9: Add path \(p'\) to the end of path \(p\)
10: \(P \leftarrow P \setminus \{p'\}\)
11: if \(c(p) \geq n\) then
12: \(stop = \text{TRUE}\)
13: end if
14: end while
15: if \(stop = \text{FALSE} \text{ and } changed = \text{TRUE}\) then
16: \((G^{\text{level}}, n) = \text{GenerateProblemInstance}(P)\)
17: \(level = level + 1\)
18: end if
19: end for
20: **output:** A sequence of instances \((G^0, n), (G^1, n), \ldots, (G^d, n)\)

instances can directly be seen as a solution to any of the other instances. The contraction mechanism is shown in Algorithm 5.

4.2 Application of ACO in the multi-level framework

The application of the ACO algorithm (see Section 3.2) in the multi-level framework works as follows. Given the sequence \((G^0, n), (G^1, n), \ldots, (G^d, n)\) of problem instances, ACO is first applied to the smallest instance \((G^d, n)\). Subsequently, ACO is applied in the given order to all problem instances \((G^{d-1}, n), \ldots, (G^0, n)\). Hereby we always use the best solution of the ACO algorithm found for an instance \((G^{r-1}, n)\) as first best-so-far solution for the application of ACO to the instance \((G^r, n)\). As stopping condition for the whole procedure we use a CPU time limit. The given CPU time is distributed such that the application of ACO to an instance \((G^r, n)\) is always allocated twice the CPU time that is allocated for the application of ACO to instance \((G^{r-1}, n)\). Due to the fact that instance \((G^{r-1}, n)\) is smaller than instance \((G^r, n)\) it is reasonable to allocate more time to \((G^r, n)\). For the application of ACO to an instance \((G^r, n)\) we use two stopping conditions: (1) the allocated CPU time, and (2) a maximum number of iterations without improving the best-so-far solution. Whenever one of the two conditions is fulfilled the application of ACO at the corresponding level is terminated, and the application to the next level starts. Note that the use of the second stopping condition implies that the last application of ACO (that is, the application to the original instance \((G^0, n)\)) may use all the remaining CPU time, which is sometimes more than the allocated CPU time. Moreover, the second stopping condition is not used for the last application of ACO. Finally, for all the experiments outlined in the following section we have set the maximum number of
iterations without improvement to 100. This value was determined by tuning by hand.

5 Experimental evaluation of the ACO approaches

We implemented our ACO approaches in ANSI C++ using GCC 3.2.2 for compiling the software. Our experimental results were obtained on a PC with an AMD64X2 4400 processor and 4 Gb of memory.

5.1 Tuning of ACO

First we performed tuning experiments in order to fix the free parameters of the ACO algorithm: the candidate list size \( \text{cls} \in \{2, 3, 5, 10, \text{all}\} \), the determinism rate \( \text{det} \in \{0.0, 0.5, 0.75, 0.9, 0.95\} \), and the number of forward solutions, respectively backward solutions, \((n_f, n_b) \in \{(6,0), (3,3), (0,6)\})\). Altogether, this results in 75 different settings of the ACO algorithm. We applied ACO with all 75 settings 10 times to each of the 200 problem instances of Set1, allowing 2 seconds for each application concerning the instances with spectrum size 100 (respectively, 10, 50, 100, or 200 seconds for the bigger instances). Then, for each of the 5 instance groups we produced a summary of the results obtained by averaging over the best run (out of 10) for each of the 40 instances of the group. These summarized tuning results are shown graphically for the biggest instances (spectrum size 500) in Figure 4. Three different measures are considered in this figure: the average global similarity score (see below), the number of instances solved to optimality (out of 40), and the average computation time needed to reach the best solution found for each instance. Hereby, the global similarity score is a measure obtained by comparing the computed DNA sequences with the DNA target sequences. We used the Needleman-Wunsch algorithm for global alignment with the following parameter settings: +1 for a match of oligonucleotides, -1 for a mismatch or a gap.\(^7\)

The results in Figure 4 allow the following conclusions. When determinism is high and the candidate list is small, the algorithm can produce very good results in a short time. However, it pays off spending a little more time (by increasing the candidate list size, for example, to 10). This improves the results while maintaining short running times. Another important observation is that the setting \((n_f, n_b) = (3,3)\) (that is, using forward as well as backward ants) generally improves over only using ants of one direction. Therefore, we decided to use the following settings for all the remaining experiments: \(\text{cls} = 10, \text{det} = 0.9\), and \((n_f, n_b) = (3,3)\).

5.2 Experiments concerning Set1

We applied ACO as well as ACO in the multi-level framework (henceforth dented by ML-ACO) to each of the 200 problem instances 10 times. From the best run for each instance we produced a summary of the results averaged over the 40 instances for each of the 5 instance groups. The results of ACO are shown in Table 4. Note that the structure of this table is the same as outlined in Section 2.5.

The results show the following. ACO is the first algorithm that is able to solve all 200 problem instances to optimality, which does not mean that all produced DNA sequences are identical to the DNA target sequences (see the similarity scores). Figure 5 shows a comparison

\(^7\)Remember in this context that an optimal solution to the SBH problem does not necessarily correspond to a DNA sequence that is equal to the target sequence.
Table 4: Results of ACO for the instances by Błażewicz et al. [3] (Set1).

<table>
<thead>
<tr>
<th>Spectrum size</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average solution quality</td>
<td>80</td>
<td>160</td>
<td>240</td>
<td>320</td>
<td>400</td>
</tr>
<tr>
<td>Solved instances</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>Average similarity score (global)</td>
<td>108.40</td>
<td>208.13</td>
<td>297.78</td>
<td>401.93</td>
<td>503.60</td>
</tr>
<tr>
<td>Average similarity score (local)</td>
<td>108.70</td>
<td>208.60</td>
<td>304.98</td>
<td>403.63</td>
<td>503.93</td>
</tr>
<tr>
<td>Average computation time (sec)</td>
<td>0.14</td>
<td>1.86</td>
<td>5.09</td>
<td>15.72</td>
<td>38.33</td>
</tr>
</tbody>
</table>

Table 5: Results of ML-ACO for the instances by Błażewicz et al. [3] (Set1).

<table>
<thead>
<tr>
<th>Spectrum size</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average solution quality</td>
<td>80</td>
<td>160</td>
<td>240</td>
<td>320</td>
<td>400</td>
</tr>
<tr>
<td>Solved instances</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>Average similarity score (global)</td>
<td>108.40</td>
<td>208.35</td>
<td>301.05</td>
<td>403.45</td>
<td>503.60</td>
</tr>
<tr>
<td>Average similarity score (local)</td>
<td>108.70</td>
<td>208.68</td>
<td>306.05</td>
<td>403.85</td>
<td>503.93</td>
</tr>
<tr>
<td>Average computation time (sec)</td>
<td>0.005</td>
<td>0.41</td>
<td>0.41</td>
<td>4.97</td>
<td>7.85</td>
</tr>
</tbody>
</table>

of the results of ACO with the results of the best metaheuristics from the literature. Hereby, EA1, EA2, and EA4 are evolutionary algorithms proposed in [8, 6], respectively [16] and [13]. TS is a tabu search approach and TS/SS a tabu search approach combined with scatter search proposed in [5]. The results show that only EA2 produces DNA sequences with similarly high global similarity scores. Concerning the number of instances solved to optimality, ACO is clearly superior to the other approaches. However, note that this measure is not given for EA2 in the literature.

Finally, we also applied ML-ACO (in the same way as ACO) to all 200 problem instances. The results are shown in Table 5. ML-ACO also solves all problem instances to optimality. Moreover, the obtained average similarity scores are comparable to the ones produced by ACO. The difference is in the computation time. The application of ACO in the multi-level framework substantially reduces the computation time. More in detail, the computation times are up to 28 times lower (concerning the smallest problem instances). In the worst case (see problem instances with spectrum size 400), the computation times are about 3 times lower.

In order to show the effects of the multilevel framework we plotted the sizes of the lower level instances concerning the 40 problem instances of spectrum size 500. The results are shown in Figure 6(a). The graphic shown in this figure provides for each level (starting from the first level below the original instance: level 1) the distribution of the spectrum sizes in form of a box plot. It is interesting to see that the first 2 contraction steps produce spectra of much reduced sizes. For example, the sizes of the spectra in level 1 are less than 40 % of the original spectra sizes. In the last contraction steps the size reduction is rather moderate.

Furthermore, we studied for each application of ML-ACO the level in which the tackled problem instance was solved. The results of this study are shown graphically in Figure 6(b) concerning the instances of spectrum size 500. Remember that we applied ML-ACO 10 times
to each of these problem instances. Out of these 400 applications, 386 were successful, which means that the tackled instance was solved to optimality. The results show that around 250 applications ended with a success already in the lowest level. This implies a very short computation time for these applications, and explains the computation time advantage of ML-ACO over ACO.

5.3 Experiments concerning Set2

Finally, we applied ACO, ML-ACO, as well as SM-FB-GREEDY(LAG) to the second set of problem instances that was introduced by Fernandes and Ribeiro in [17] (i.e., Set2). In contrast to Set1, Set2 consists of larger problem instances, and of problem instances whose spectra are comprised of probes of different lengths (i.e., $l \in \{7, 8, 9, 10\}$). The results concerning the average global similarity scores (in %) are shown in Figure 7, whereas the results concerning the computing times are shown in Figure 8. They allow us to draw the following conclusions:

1. As expected, the results of all three algorithms are much better when the probes are longer. In fact, while ACO and ML-ACO achieve average global similarity scores of around 98% even for the largest problem instances when $l = 10$, their performance drops to a value around 58% for the largest problem instances when $l = 7$.

2. Concerning the comparison between the ACO approaches and SM-FB-GREEDY(LAG), we can note that the ACO approaches are in general clearly better. However, this difference decreases with decreasing $l$. An advantage of SM-FB-GREEDY(LAG) is clearly the computation time, which is around 1 second for the largest problem instances. In contrast, the computation times of the ACO approaches increase at a faster rate with increasing problem size.

3. When comparing ACO with ML-ACO, the graphics show that ML-ACO has in general a slight advantage over ACO in terms of the average global similarity scores. In terms of computation times, ML-ACO has a strong advantage over ACO when the probe length is high. However, with decreasing probe length this advantage turns into a disadvantage. The reason is that when the probe length is low the instance contraction mechanism is much more error prone, and instances can often not be solved in low levels. Therefore, ML-ACO basically wastes computation time by trying to solve instances in low levels.

6 Conclusions and outlook to the future

In this work we have proposed new constructive heuristics for the problem of DNA sequencing by hybridization. First, we extended an existing heuristic that produces solutions sequentially. Then, we proposed a conceptionally new heuristic that is based on merging shorter DNA strands into bigger ones until a DNA strand of sufficient size is obtained. The best results were obtained by hybrid heuristics that combine the strengths of both heuristic types. They are comparable to the results of existing metaheuristic methods. However, the constructive heuristics need fewer computation time.

Based on our constructive heuristics we also presented an ant colony optimization algorithm for DNA sequencing by hybridization. Moreover, we presented a so-called multilevel framework in which the ant colony optimization algorithm can be applied. The results show
that the proposed ant colony techniques are state-of-the-art methods for popular problem instances proposed in the literature.

Future work consists in the application (and adaptation) of our techniques to larger problem instances, because in practise biologists are often faced with DNA sequences of several 10000 nucleotide bases. Preliminary tests show that our hybrid constructive heuristics needs around 100 seconds of computation time for the application to a spectrum created from a target sequence of length 10000 and probe size 10. However, the ACO approaches are clearly not applicable to such problem instances. One possibility consists in constricting the search of the ACO algorithm to the lowest level instance produced by the contraction mechanism. However, we are also studying other possibilities.

Acknowledgements

The authors would like to thank M. Kasprzak and Eraldo R. Fernandes for providing benchmark instance sets.

References


Figure 3: Comparison of SM-FB-GREEDY(LAG) with meta-heuristics from the literature. The comparison is done concerning the global average similarity score obtained for the instances by Błażewicz et al. [3]. Note that EA3 is not included because from [12] it is not clear which alignment algorithm was used by the authors.
(a) Tuning results for \( n_f = 6 \) (that is, 6 forward solutions per iteration), and \( n_b = 0 \) (that is, no backward solutions per iteration).

(b) Tuning results for \( n_f = 3 \) and \( n_b = 3 \).

(c) Tuning results for \( n_f = 0 \) and \( n_b = 6 \).

Figure 4: Tuning results of ACO for the 40 instances with spectrum size 500. Each square of the 9 5x5 matrices corresponds to one algorithm setting. The matrix rows correspond to the 5 values of the candidate list size (that is (from top to bottom), \( cls \in \{2, 3, 5, 10, \text{all}\} \)), and the columns correspond to the determinism rate (that is (from left to right), \( q \in \{0.0, 0.5, 0.75, 0.9, 0.95\} \)). The first matrix of each subfigure corresponds to the values of the global similarity score, the second matrix shows the number of instances solved to optimality, and the third matrix visualizes the computation times. In the three left most matrices (showing together the 75 different settings of ACO), 400.1 was the lowest global similarity score obtained by any setting. This score corresponds to black color. The highest score obtained is 503.6, which corresponds to white color. This is similar for the 3 matrices on the right visualizing the computation times: 29.845 seconds corresponds to black color, and 178.454 seconds to white color. All the remaining values are shown in corresponding grey scale.
Figure 5: Comparison of ACO with the best metaheuristics from the literature concerning (a) the global average similarity score obtained, and (b) the number of instances solved to optimality. The comparison concerns the instances of Blažewicz et al. [3]. Note that for EA2 the number of solved instances is not given in the literature.
Figure 6: Results concerning studies of the multilevel framework.
Figure 7: Comparison of ACO, ML-ACO, and our best heuristic SM-FB-GREEDY(LAG) on instances whose spectra are composed of probes (i.e., oligonucleotides) of different length. The comparison concerns the average global similarity score (in %). The instances were generated by Fernandes and Ribeiro in [17].
Figure 8: Computation time comparison of ACO, ML-ACO, and our best heuristic SM-FB-GREEDY(LAG) on instances whose spectra are composed of probes (i.e., oligonucleotides) of different length. The instances were generated by Fernandes and Ribeiro in [17].