Available online at www.sciencedirect.com

ScienceDirect

Computers & Operations Research 35 (2008) 3620–3635

computers & operations research

www.elsevier.com/locate/cor

An ant colony optimization algorithm 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

Available online 27 March 2007

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. Several metaheuristics such as tabu search and evolutionary algorithms have been applied to this problem. However, the performance of existing metaheuristics is often inferior to the performance of recently proposed constructive heuristics. On the basis of these new heuristics we develop an ant colony optimization algorithm for DNA sequencing by hybridization. An important feature of this algorithm is the implementation in a so-called multi-level framework. The computational results show that our algorithm is currently a state-of-the-art method for the tackled problem.
© 2007 Elsevier Ltd. All rights reserved.

Keywords: Ant colony optimization; Multi-level techniques; 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 *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 1980s, an alternative and much faster method called *DNA sequencing by hybridization* was developed (see [2–4]).

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 a certain length l. For example, consider a DNA array of all possible

☆ Note that a preliminary version of this paper was published in the proceedings of HM 2006 [1].
* Corresponding author.
E-mail addresses: cblum@lsi.upc.edu (C. Blum), mateuyabar@gmail.com (M.Y. Vallès), mjblesa@lsi.upc.edu (M.J. Blesa).

0305-0548/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2007.03.007

probes of length $l = 3$ (see also [5]):

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

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 [6]. 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 with optimal solutions that can be shown to have a high probability to resemble the target sequence. In this work we consider the one proposed by Błażewicz et al. in [7].[1]

Henceforth, let the target sequence be denoted by s_t. The number of nucleotide bases of s_t shall be denoted by n, that is, $s_t \in \{A, C, G, T\}^n$. 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 (a short DNA strand) of length l, that is, $i \in \{A, C, G, T\}^l$. Let $G = (V, A)$ be the completely connected directed graph defined by $V = S$ (see also [5]). 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). Each directed, node-disjoint path $p = (i_1, \ldots, i_k)$ in G is a solution to the problem. The number of vertices (i.e., oligonucleotides) on a path p is called the length of the path, denoted by $\text{len}(p)$. In the following we denote by $p[r]$ the rth vertex on a given path p (starting from position 1). In contrast to the length, the cost of a path p is defined as follows:

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

[1] In [8] it was shown that this model is *NP*-hard.

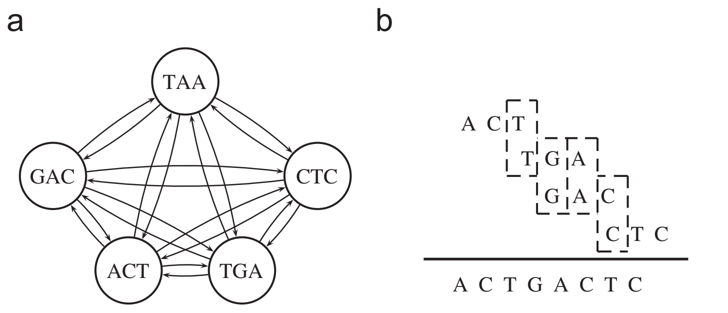
a
b
TAA
GAC
CTC
ACT
TGA
A C T
T G A
G A C
C T C
A C T G A C T C
Completely connected directed graph G.
DNA sequence retrieval from a solution.

Fig. 1. Example. (a) The completely connected directed graph G with the spectrum $S = \{\text{ACT, TGA, GAC, CTC, TAA}\}$ as 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 solution is $p^* = (\text{ACT, TGA, GAC, CTC})$. 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.

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 solution (or path) p^* in G with $\text{len}(p^*) \geqslant \text{len}(p)$ for all possible solutions p that fulfill $c(p) \leqslant n$. In the following we refer to this optimization problem as *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 solution in this example is $p^* = (\text{ACT, TGA, GAC, CTC})$ with $\text{len}(p^*) = 4$ and $c(p^*) = 8$. The DNA sequence that is retrieved from this path is equal to the target sequence (see Fig. 1).

1.3. Existing approaches

The first approach to solve the SBH problem was a branch and bound method proposed in [7]. 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 1 h. Another argument against this branch and 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 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, [9]). For an overview on the existing approaches for the SBH problem see Table 1.[2]

[2] Note that the GRASP method proposed in [18] 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

Type of algorithm	Identifier	Publication
Constructive heuristic	LAG	Błażewicz et al. [7], 1999
Constructive heuristic	OW	Błażewicz et al. [10], 2002
Evolutionary algorithm	EA1	Błażewicz et al. [11,12], 2002
Evolutionary algorithm	EA2	Endo [13], 2004
Evolutionary algorithm	EA3	Brizuela et al. [14], 2004
Evolutionary algorithm	EA4	Bui and Youssef [15], 2004
Tabu search	TS	Błażewicz et al. [16], 2000
Tabu search/scatter search hybrid	TS/SS	Błażewicz et al. [17,12], 2004
GRASP-like multi-start technique	GRASP	Fernandes and Ribeiro [18], 2005

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 Błażewicz et al. in [16].[3] This instance set consists of 40 real DNA target sequences of length 109, 209, 309, 409, and 509 (200 instances in total). 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 [18]. 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.

1.5. Organization of the paper

The organization of the paper is as follows. In Section 2 we present the constructive heuristic that will be used by our ant colony algorithm for the probabilistic construction of solutions. In Section 3 we present the ant colony optimization approach, whereas in Section 4 we propose a multi-level framework in which our algorithm can be applied. Finally, in Section 5 we conduct an experimental evaluation of our algorithms, comparing them to most of the techniques from the literature. In Section 6 we offer conclusions and an outlook to future work.

2. A simple constructive heuristic

For the description of the constructive heuristic we use the following notation:

$$\text{pre}(i) := \text{argmax}\{o_{j\,i} \mid j \in \hat{S}, j \neq i\}, \tag{2}$$

$$\text{suc}(i) := \text{argmax}\{o_{i\,j} \mid j \in \hat{S}, j \neq i\}, \tag{3}$$

where $\hat{S} \subseteq S$ and $i \in \hat{S}$ are given. In words, $\text{pre}(i)$ 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, $\text{suc}(i)$ is the best available successor for i in $\hat{S}$. In case of ties, the first one that is found is taken.

Given a problem instance (G, n), a 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

[3] The problem instances are available from http://bio.cs.put.poznan.pl/index.php?id = 62.

function Choose_Initial_Oligonucleotide(S). In subsequent construction steps p is extended by adding exactly one oligonucleotide chosen in function Choose_Next$(\hat{S})$. Finally, the solution construction stops as soon as $c(p) \geqslant n$, that is, when 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 Find_Best_Subpath(p) searches for the longest (in terms of the number of oligonucleotides) subpath p' of p such that $c(p') \leqslant n$, and replaces p by p'.

Algorithm 1 (*The* GREEDY *heuristic*).

```
1:  input: A problem instance $(G, n)$
2:  $i^* :=$ Choose_Initial_Oligonucleotide$(S)$
3:  $p := (i^*)$
4:  $\hat{S} := S$
5:  while $c(p) < n$ do
6:    $\hat{S} := \hat{S} \backslash \{i^*\}$
7:    $i^* :=$ Choose_Next$(\hat{S})$
8:    Extend path $p$ by adding $i^*$ to its end
9:  endwhile
10: $p :=$ Find_Best_Subpath$(p)$
11: output: DNA sequence $s$ that is obtained from $p$
```

Different versions of GREEDY are obtained by different implementations of the functions Choose_Initial_Oligonucleotide(S) and Choose_Next$(\hat{S})$. The first and only version that was published in the literature so far (algorithm LAG in Table 1) implements these functions as follows. The function Choose_Initial_Oligonucleotide(S) chooses an oligonucleotide uniformly at random from S. Moreover, function Choose_Next$(\hat{S})$ chooses the oligonucleotide i^* such that

$$i^* := \text{argmax}\{o_{p[\text{len}(p)]i} + o_{i\,\text{suc}(i)} \mid i \in \hat{S}\}, \tag{4}$$

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

In [19] we presented an improved version of this heuristic—denoted by GREEDY(LAG)—using the following implementation of function Choose_Initial_Oligonucleotide(S). First, $S_{bs} \subset S$ 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

$$S_{bs} := \{i \in S \mid o_{i\ \text{suc}(i)} \geqslant o_{j\ \text{suc}(j)}, \forall j \in S\}. \tag{5}$$

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

$$S_{wp} := \{i \in S_{bs} \mid o_{\text{pre}(i)\ i} \leqslant o_{\text{pre}(j)\ j}, \forall j \in S_{bs}\}. \tag{6}$$

Note that the best predecessors pre(i) of oligonucleotides $i \in S_{bs}$ are determined with respect to S. 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.

In ant colony optimization algorithms we need a construction mechanism that produces reasonably good solutions in a short computation time. Therefore, we tested also a simplified version of GREEDY(LAG) (henceforth denoted by GREEDY(S)) that does not use the look-ahead strategy. The implementation of function Choose_Next$(\hat{S})$ in GREEDY(S) is as follows. Oligonucleotide i^* is chosen such that $i^* := \text{suc}(p[\text{len}(p)])$.

2.1. Results of the constructive heuristics

We implemented the two constructive heuristics in ANSI C + + and used 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. We applied the constructive heuristics to all problem instances of benchmark set Set1 (see Section 1.4). The results are shown

Table 2
Results of the constructive heuristics for benchmark instances of Set1, that is, the instances by Błażewicz et al. [7]

Spectrum size	100	200	300	400	500
(a) *Results of* GREEDY(S)					
Average solution quality	77.20	152.78	229.00	302.73	375.28
Solved instances	26	18	18	7	5
Average similarity score (global)	80.85	142.10	199.98	214.05	269.68
Average similarity score (local)	95.28	167.20	234.80	277.33	334.90
Average computation time (sec)	0.0028	0.012	0.025	0.044	0.069
(b) *Results of* GREEDY(LAG)					
Average solution quality	76.98	153.53	230.68	309.03	383.08
Solved instances	23	15	12	7	4
Average similarity score (global)	77.05	133.63	171.78	206.80	218.60
Average similarity score (local)	91.83	152.43	209.33	272.40	293.48
Average computation time (sec)	0.0035	0.016	0.037	0.076	0.13

numerically in Table 2. 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.[4] 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. Surprisingly, the results of GREEDY(S) are in general not worse than the results of GREEDY(LAG). In particular, GREEDY(S) even seems to have slight advantages over GREEDY(LAG) in terms of similarity scores, and slight disadvantages in terms of solution quality. This suggests that the look-ahead strategy is not necessary for this type of constructive heuristic. This is good news, because the simple version only spends about half of the computation time the look-ahead version spends.

3. An ant colony optimization approach

In this section we present an ant colony optimization (ACO) algorithm [20] 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 $\mathscr{T}$), 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.

The key component of any ACO algorithm is the underlying heuristic for constructing solutions to the tackled optimization problem. Generally, ACO algorithms can be very successful if they are based on a well-working constructive heuristic. For the SBH problem we have given well-working alignment heuristics such as the one presented in Algorithm 1. This was our motivation for developing an ACO approach for the SBH problem.

[4] 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.

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),[5] the original objective function value $\text{len}(p)$ is the number of oligonucleotides in p. This objective function has the following disadvantage when used in a stochastic search algorithm. Let p and p' be two solutions with $\text{len}(p) = \text{len}(p')$ and $c(p) < c(p')$.[6] Even though the objective function $\text{len}(\cdot)$ cannot 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 $c(p) \leqslant n$. Therefore, we define a comparison operator $f(\cdot)$ as follows:

$$f(p) > f(p') \quad \Leftrightarrow \quad \text{len}(p) > \text{len}(p') \quad \text{or} \quad (\text{len}(p) = \text{len}(p') \text{ and } c(p) < c(p')). \tag{7}$$

3.2. The algorithm

Our ACO approach, which is a $\mathcal{MAX}$–$\mathcal{MIN}$ ant system ($\mathcal{MM}$AS) implemented in the hyper-cube framework (HCF) [21], solves the SBH problem as shown in Algorithm 2. The data structures used by this algorithm, in addition to counters and to the pheromone model $\mathcal{T}$, are:

- the *iteration-best* solution p_{ib}: the best solution generated in the current iteration by the ants;
- the *best-so-far* solution p_{bs}: the best solution generated since the start of the algorithm;
- the *restart-best* solution p_{rb}: the best solution generated since the last restart of the algorithm;
- the *convergence factor cf*, $0 \leqslant cf \leqslant 1$: 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 0.5 in procedure InitializePheromoneValues($\mathcal{T}$). At each iteration, first n_f ants construct a solution each in procedure ConstructForwardSolution($\mathcal{T}$), and then n_b ants construct a solution each in procedure ConstructBackwardSolution($\mathcal{T}$). 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 ApplyPheromoneUpdate(*cf*, *bs_update*, $\mathcal{T}$, 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 ResetPheromoneValues($\mathcal{T}$) 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 2 are now described in detail.

Algorithm 2 (*ACO for the SBH problem*).

```
1:  input: a problem instance $(G, n)$
2:  $p_{bs} :=$ NULL, $p_{rb} :=$ NULL, $cf := 0$, $bs\_update :=$ FALSE
3:  InitializePheromoneValues($\mathcal{T}$)
4:  while termination conditions not satisfied do
5:      for $j = 1$ to $n_f$ do
6:          $p_j :=$ ConstructForwardSolution($\mathcal{T}$)
7:      end for
```

[5] Remember 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 directed, node-disjoint path in G.

[6] Remember that $c(p)$ denotes the length of the DNA sequence derived from p. See Fig. 1(b) for an example.

```
8:     for $j = (n_f + 1)$ to $(n_f + n_b)$ do
9:       $p_j :=$ ConstructBackwardSolution($\mathscr{T}$)
10:    end for
11:    $p_{ib} :=$ argmax($f(p_1), \ldots, f(p_{n_f+n_b})$)
12:    if $p_{rb} =$ NULL or $f(p_{ib}) > f(p_{rb})$ then $p_{rb} := p_{ib}$
13:    if $p_{bs} =$ NULL or $f(p_{ib}) > f(p_{bs})$ then $p_{bs} := p_{ib}$
14:    ApplyPheromoneUpdate($cf$, $bs\_update$, $\mathscr{T}$, $p_{ib}$, $p_{rb}$, $p_{bs}$)
15:    $cf :=$ ComputeConvergenceFactor($\mathscr{T}$)
16:    if $cf > 0.9999$ do
17:      if $bs\_update =$ TRUE then
18:        ResetPheromoneValues($\mathscr{T}$)
19:        $p_{rb} :=$ NULL
20:        $bs\_update :=$ FALSE
21:      else
22:        $bs\_update :=$ TRUE
23:      end if
24:    end if
25:  end while
26:  output: $p_{bs}$
```

ConstructForwardSolution($\mathscr{T}$): This function constructs a directed, node-disjoint path $p = (i_1, \ldots, i_k)$ in G from left to right by using a probabilistic version of the GREEDY heuristic (see Algorithm 1). Both functions Choose_Initial_Oligonucleotide(S) and Choose_Next($\hat{S}$) of Algorithm 1 utilize a pheromone model $\mathscr{T}$, which consists of pheromone values τ_{ij} and τ_{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, $\mathscr{T}$ comprises pheromone values τ_{0i} and τ_{i0} for all $i \in S$, where 0 is a non-existing dummy oligonucleotide.

Function Choose_Next($\hat{S}$) is implemented as follows. First, a desirability value $\mu_{i_t j} := \tau_{i_t j} \cdot [\eta_{i_t j}]^5$ is computed for all $j \in \hat{S}$, where $\eta_{i_t j} := o_{i_t j}/(l-1)$. The values $\eta_{i_t j}$ are called *heuristic information*. They are defined such that $\eta_{i_t j} \in [0, 1]$ grows with growing overlap $o_{i_t j}$ between the oligonucleotides i_t and j. Note that when the pheromone values are all equal, the desirability value $\mu_{i_t j}$ is high exactly when $o_{i_t j}$ is high. Then, we generate a so-called *restricted candidate list* $S^{\text{rcl}} \subseteq \hat{S}$ with a pre-defined cardinality *cls* such that $\mu_{i_t j} \geqslant \mu_{i_t u}$ for all $u \in \hat{S}$ and $j \in S^{\text{rcl}}$. Then, with probability $q \in [0, 1)$ the next oligonucleotide i_{t+1} is chosen from S^{rcl} such that

$$i_{t+1} := \arg \max_{j \in S^{\text{rcl}}} \{\mu_{i_t j}\}. \tag{8}$$

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

$$\mathbf{p}_{i_t j} := \frac{\mu_{i_t j}}{\sum_{u \in S^{\text{rcl}}} \mu_{i_t u}}. \tag{9}$$

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

In contrast to function Choose_Next($\hat{S}$), in function Choose_Initial_Oligonucleotide(S) the desirability values are computed as $\mu_{0j} := \tau_{0j} \cdot [\eta_{0j}]^5 \in [0, 1]$ for all $j \in \hat{S}$ (note that $\hat{S} = S$ when p is empty). Hereby,

$$\eta_{0j} := \frac{l - o_{\text{pre}(j)\, j} + o_{j\, \text{suc}(j)}}{2(l-1)}. \tag{10}$$

Note that this way of defining the heuristic information favors 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

Table 3
Setting of κ_{ib}, κ_{rb} and κ_{bs} depending on the convergence factor *cf* and the Boolean control variable *bs_update*

	$bs\_update = \text{FALSE}$				$bs\_update = \text{TRUE}$
	$cf < 0.7$	$cf \in [0.7, 0.9)$	$cf \in [0.9, 0.95)$	$cf \geqslant 0.95$	
κ_{ib}	1	2/3	1/3	0	0
κ_{rb}	0	1/3	2/3	1	0
κ_{bs}	0	0	0	0	1

candidate list S^{rcl} and the choice of one of the oligonucleotides from S^{rcl} is the same as outlined above for function Choose_Next($\hat{S}$).

ConstructBackwardSolution($\mathscr{T}$): This function works principally in the same way as does function ConstructForwardSolution($\mathscr{T}$). The first difference is that a solution p is constructed from right to left. The second difference is that—given a partial solution $p = (i_t, \ldots, i_1)$—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 μ_{ji_t} (instead of $\mu_{i_t j}$). 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.

ApplyPheromoneUpdate(*cf*, *bs_update*, $\mathscr{T}$, p_{ib}, p_{rb}, p_{bs}): as usual for $\mathscr{MM}$AS 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:

$$\tau_{ij} \leftarrow \tau_{ij} + \rho \cdot (m_{ij} - \tau_{ij}) \quad \forall \tau_{ij} \in \mathscr{T}, \tag{11}$$

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

$$m_{ij} \leftarrow (\kappa_{ib} \cdot \delta_{ij}(p_{ib})) + (\kappa_{rb} \cdot \delta_{ij}(p_{rb})) + (\kappa_{bs} \cdot \delta(p_{bs})), \tag{12}$$

where κ_{ib} is the weight of solution p_{ib}, κ_{rb} is the weight of solution p_{rb}, κ_{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 $i = 0$, $\delta_{0j}(p)$ returns 1 in case j is the first oligonucleotide in p, and 0 otherwise. In case $j = 0$, $\delta_{i0}(p)$ returns 1 in case i is the last oligonucleotide in p, and 0 otherwise. After the pheromone update rule (Eq. (11)) 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.

Eq. (12) 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 $\mathscr{MM}$AS 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., $0.7 \leqslant cf < 0.9$) one third of the total influence is given to the restart-best solution, which then increases to two thirds when $0.9 \leqslant cf < 0.95$. Eventually, all the influence is given to the restart-best solution (i.e., when $cf \geqslant 0.95$). 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.

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

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

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 [22]) 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 an 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

The first step of a multi-level framework consists in contracting the original problem instance iteratively in order to generate a sequence of smaller and smaller problem instances. The idea for the SBH problem, which stems from a constructive heuristic from [19], is as follows: we successively try to identify growing sub-paths in G that are most probably parts of optimal solutions. Such a set of sub-paths can be used to produce a smaller graph by merging all the nodes of a sub-path into one single node. More in detail, we used the following mechanism (see Algorithm 3): in the initial situation we have given graph G, and a set P of $|S|$ paths, each of which contains exactly one oligonucleotide $i \in S$. A contraction step consists of merging some of these paths. Hereby, we consider only those paths p and p' in which the last oligonucleotide of p and the first one of p' have a pre-defined overlap. The setting of the pre-defined overlap starts with the maximum $l-1$ and is being reduced step by step. In addition it is required that p' is the unique best successor of p, and that p is the unique best predecessor of p'. The best successor (respectively, predecessor) of a path p are defined as follows:

$$\mathrm{suc}(p) := \mathrm{argmax}\{o_{p\,p'} \mid p' \in P, p' \neq p\}, \tag{13}$$

$$\mathrm{pre}(p) := \mathrm{argmax}\{o_{p'\,p} \mid p' \in P, p' \neq p\}. \tag{14}$$

Hereby, $o_{p\,p'}$ is defined as the overlap between the last oligonucleotide of p and the first one of p'. Furthermore, in Algorithm 3 $S_{\mathrm{suc}}(p)$ is defined as the set of best successors of p, that is, $S_{\mathrm{suc}}(p) := \{p' \in P \mid o_{p\,p'} = o_{p\,\mathrm{suc}(p)}\}$; and $S_{\mathrm{pre}}(p)$ is defined as the set of best predecessors of p, that is, $S_{\mathrm{pre}}(p) := \{p' \in P \mid o_{p'\,p} = o_{\mathrm{pre}(p)\,p}\}$.

Algorithm 3 (*Instance contraction*).

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

Each contraction 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 Fig. 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) refers to the smallest instance that can be obtained (that is, a further construction step would produce a path p with $c(p) \geqslant n$). Note that all these problem instances have the same target sequence. Moreover, a solution to any of these instances can directly be seen as a solution to any of the other instances.

4.2. *Application of ACO in the multi-level framework*

The application of the ACO algorithm 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. First, we performed tuning experiments in order to fix the free parameters of the ACO algorithm: the candidate list size $cls \in \{2, 3, 5, 10, \text{all}\}$, the determinism rate $q \in \{0.0, 0.5, 0.75, 0.9, 0.95\}$, and the number of forward solutions, respectively, backward solutions, $(n_f, n_b) \in \{(6, 0), (3, 3), (0, 6)\}$. This results in 75 different settings of the ACO algorithm. The tuning results are presented in detail in [1]. Here, we only give a summary: When the determinism is high and the candidate lists are 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) significantly improves over only using ants of one direction, that is, the setting (3, 3) results in higher quality solutions obtained in a shorter amount of time (see Fig. 2). Therefore, we decided to use the following settings for all our experiments: $cls = 10$, $q = 0.9$, and $(n_f, n_b) = (3, 3)$.

5.1. *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 five 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 2.1.

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). Fig. 3 shows a comparison of the results of ACO with the results of the best metaheuristics from the literature.[7]

[7] The algorithm notifiers are explained in Table 1.

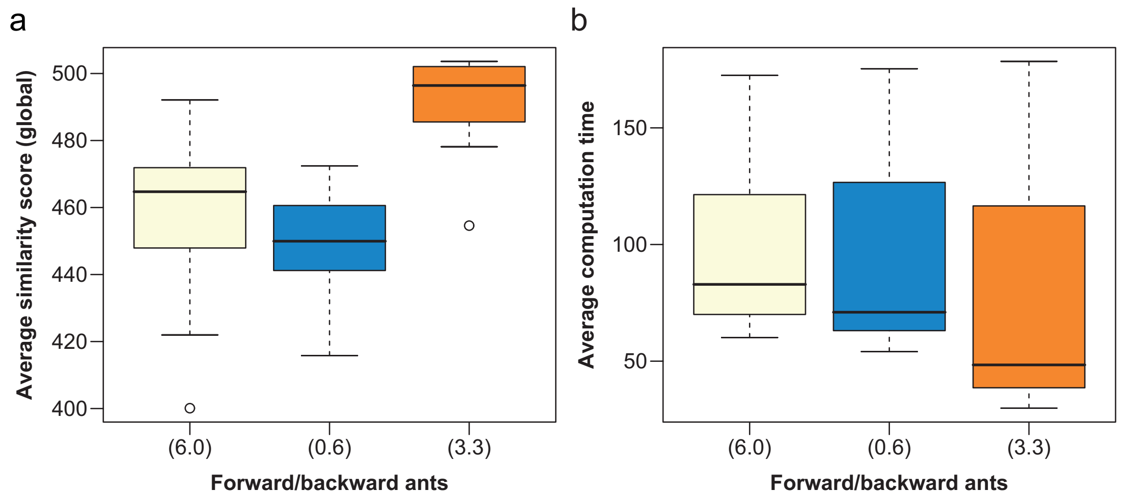

Fig. 2. Tuning results. (a) shows for each of the three settings of (n_f, n_b) the distribution of the results (the average similarity scores) obtained by the ACO algorithms with the 25 different settings concerning *cls* and *q*. The distributions are shown in the form of box-plots. (b) shows the corresponding distributions of the computation times.

Table 4
Results of ACO for the instances by Błażewicz et al. [7] (Set1)

Spectrum size	100	200	300	400	500
Average solution quality	80	160	240	320	400
Solved instances	40	40	40	40	40
Average similarity score (global)	108.40	208.13	297.78	401.93	503.60
Average similarity score (local)	108.70	208.60	304.98	403.63	503.93
Average computation time (s)	0.14	1.86	5.09	15.72	38.33

Table 5
Results of ML-ACO for the instances by Błażewicz et al. [7] (Set1)

Spectrum size	100	200	300	400	500
Average solution quality	80	160	240	320	400
Solved instances	40	40	40	40	40
Average similarity score (global)	108.40	208.35	301.05	403.45	503.60
Average similarity score (local)	108.70	208.68	306.05	403.85	503.93
Average computation time (s)	0.005	0.41	0.41	4.97	7.85

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. In particular, 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 Fig. 4(a). The graphic shown in this figure

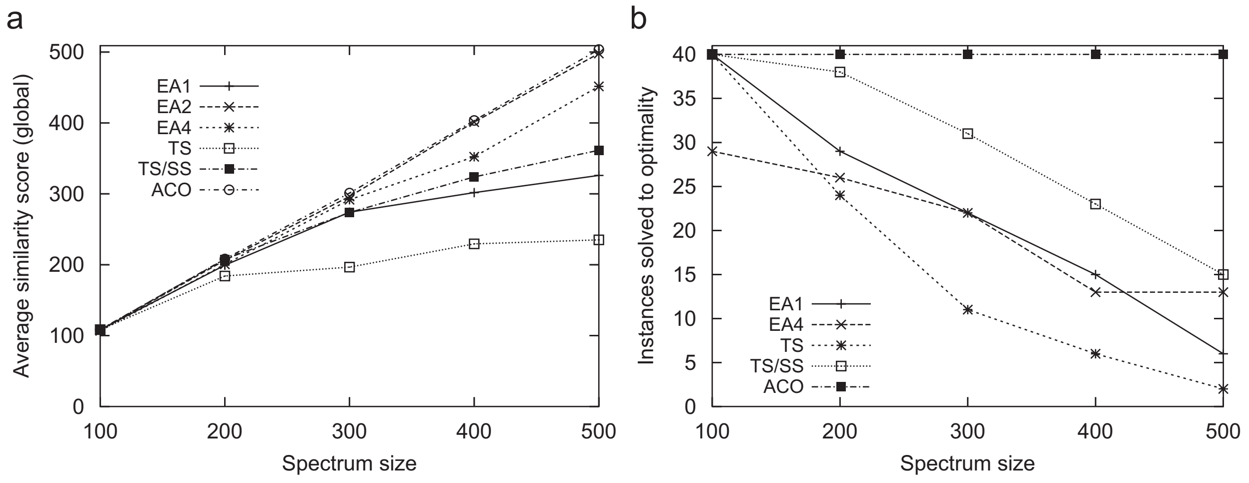

Fig. 3. Comparison of ACO with the best metaheuristics from the literature concerning (a) the global average similarity score obtained, and (b) the number of instances solved to optimality. The comparison concerns the instances of Błażewicz et al. [7]. Note that for EA2 the number of solved instances is not given in the literature.

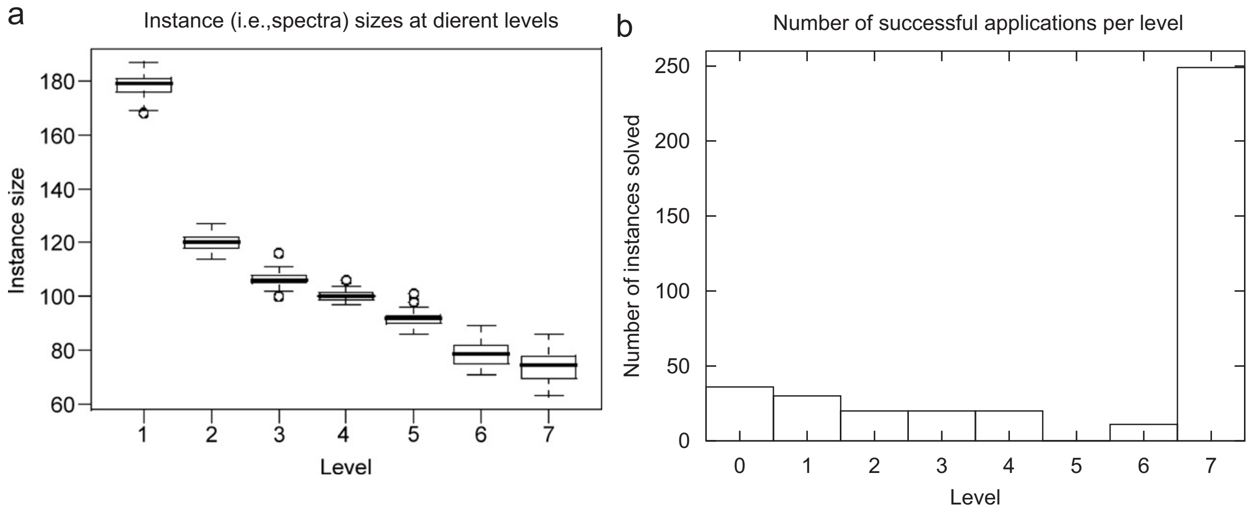

Fig. 4. Results concerning studies of the multilevel framework. (a) Instance (i.e., spectra) sizes at different levels, (b) number of successful applications per level.

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 two 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 Fig. 4(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.

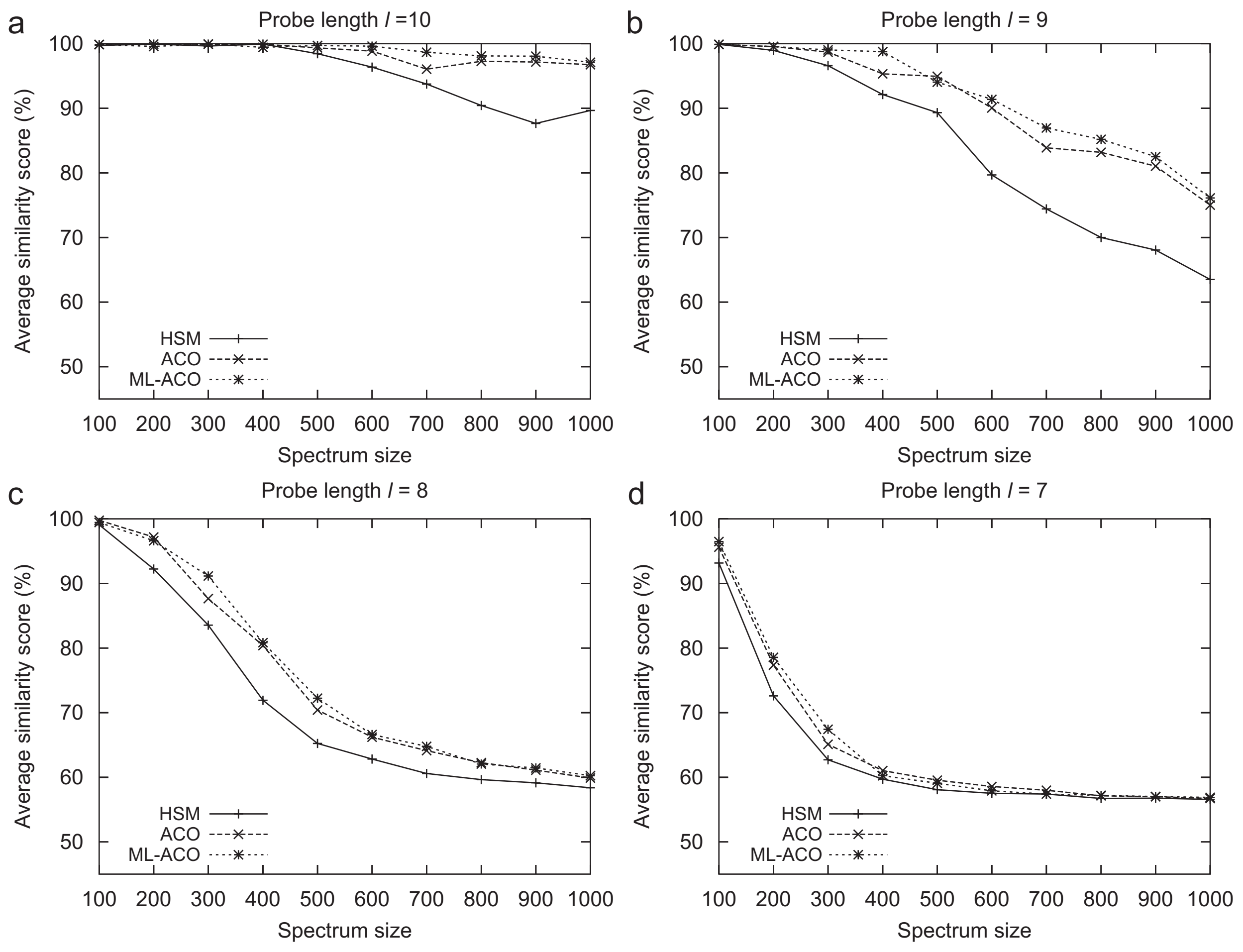

Fig. 5. Comparison of ACO, ML-ACO, and the heuristic HSM on instances whose spectra are composed of probes (i.e., oligonucleotides) of different length. The comparison concerns the average global similarity score (in %). The instances were generated by Fernandes and Ribeiro in [18]. (a) Probe length $l = 10$, (b) probe length $l = 9$, (c) probe length $l = 8$, (d) probe length $l = 7$.

5.2. *Experiments concerning* Set2

Finally, we applied ACO and ML-ACO to the second set of problem instances that was introduced by Fernandes and Ribeiro in [18] (i.e., Set2). In contrast to Set1, Set2 contains also larger problem instances, as well as problem instances whose spectra are comprised of probes of different lengths (i.e., $l \in \{7, 8, 9, 10\}$). We compared the results to the best available constructive heuristic, that is, heuristic HSM presented in [19]. This heuristic obtains results that are comparable to the results of the best available metaheuristics. The results concerning the average global similarity scores (in %) are shown in Fig. 5, whereas the results concerning the computation times are shown in Fig. 6. 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 HSM, we can note that the ACO approaches are in general clearly better. However, this difference decreases with decreasing l. An advantage of HSM is clearly the computation time, which is around 1 s for the largest problem instances. In contrast, the computation times of the ACO approaches increase at a much faster rate with increasing problem size.

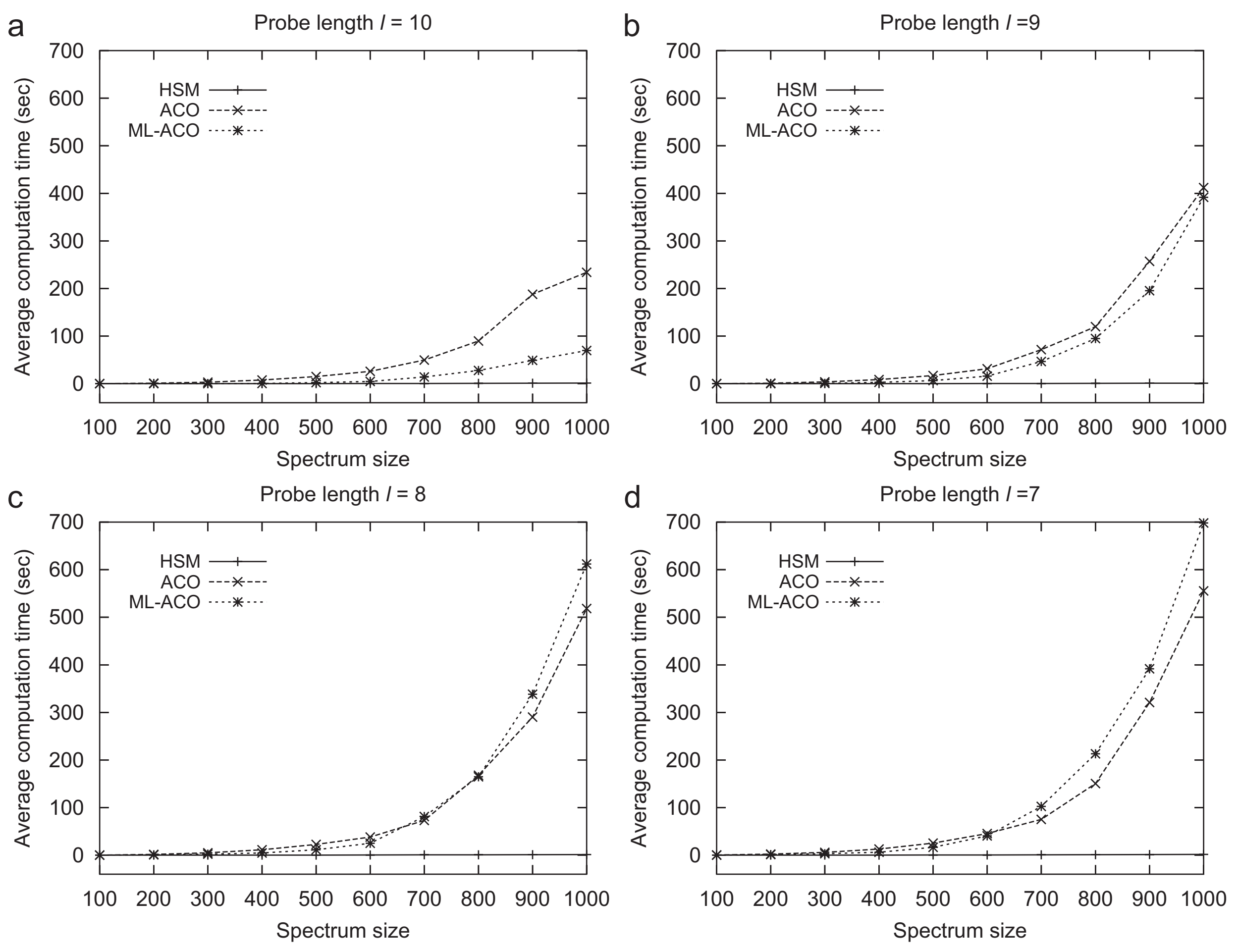

Fig. 6. Computation time comparison of ACO, ML-ACO, and the heuristic HSM on instances whose spectra are composed of probes (i.e., oligonucleotides) of different length. The instances were generated by Fernandes and Ribeiro in [18]. (a) Probe length $l = 10$, (b) probe length $l = 9$, (c) probe length $l = 8$, (d) probe length $l = 7$.

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

Based on well-working constructive heuristics from the literature we developed 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 benchmark instances from the literature.

Future work consists in the application (and adaptation) of our ACO algorithms as well as the heuristics we proposed in [19] to larger problem instances. This is important, because in practise biologists are often faced with DNA sequences of several 10 000 nucleotide bases. Preliminary tests show that the best existing constructive heuristic needs around 100 s of computation time for the application to a spectrum created from a target sequence of length 10 000 and probe size 10.

The ACO approaches are clearly not applicable to such problem instances. One possibility for making them applicable consists in restricting the search to the lowest level instance produced by the contraction mechanism. However, we are also studying other possibilities.

Acknowledgments

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

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 research fellow. Maria Blesa acknowledges support from the Spanish Ministry of Education and Science under the project CICYT TIN-2005-09198 (ASCE).

References

[1] Blum C, Yábar Vallès M. Multi-level ant colony optimization for DNA sequencing by hybridization. In: Almeida F, Blesa M, Blum C, Moreno JM, Pérez M, Roli A, Sampels M, editors. Proceedings of HM 2006—3rd International Workshop on Hybrid Metaheuristics, Lecture notes in computer science, vol. 4030. Berlin, Germany: Springer; 2006. p. 94–109.
[2] Bains W, Smith GC. A novel method for nucleic acid sequence determination. Journal of Theoretical Biology 1988;135:303–7.
[3] Lysov IuP YP, Florentiev VL, Khorlin AA, Khrapko KR, Shik VV. Determination of the nucleotide sequence of DNA using hybridization with oligonucleotides. A new method. Doklady Akademii nauk SSSR 1988;303:1508–11.
[4] Drmanac R, Labat I, Brukner R, Crkvenjakov R. Sequencing of megabase plus DNA by hybridization: theory of the method. Genomics 1989;4:114–28.
[5] Idury RM, Waterman MS. A new algorithm for DNA sequence assembly. Journal of Computational Biology 1995;2(2):291–306.
[6] Pevzner PA. l-tuple DNA sequencing: computer analysis. Journal of Biomolecular Structure and Dynamics 1989;7:63–73.
[7] Błażewicz J, Formanowicz P, Kasprzak M, Markiewicz WT, Weglarz J. DNA sequencing with positive and negative errors. Journal of Computational Biology 1999;6:113–23.
[8] Błażewicz J, Kasprzak M. Complexity of DNA sequencing by hybridization. Theoretical Computer Science 2003;290(3):1459–73.
[9] Blum C, Roli A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Computing Surveys 2003;35(3): 268–308.
[10] Błażewicz J, Formanowicz P, Guinand F, Kasprzak M. A heuristic managing errors for DNA sequencing. Bioinformatics 2002;18(5):652–60.
[11] Błażewicz J, Kasprzak M, Kuroczycki W. Hybrid genetic algorithm for DNA sequencing with errors. Journal of Heuristics 2002;8:495–502.
[12] Błażewicz J, Glover F, Kasprzak M. Evolutionary approaches to DNA sequencing with errors. Annals of Operations Research 2005;138: 67–78.
[13] Endo TA. Probabilistic nucleotide assembling method for sequencing by hybridization. Bioinformatics 2004;20(14):2181–8.
[14] Brizuela CA, González LC, Romero HJ. An improved genetic algorithm for the sequencing by hybridization problem. In: Raidl GR, Cagnoni S, Branke J, Corne D, Drechsler R, Jin Y, Johnson CG, Machado P, Marchiori E, Rothlauf F, Smith GD, Squillero G, editors. Proceedings of the EvoWorkshops—Applications of Evolutionary Computing: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC, Lecture notes in computer science, vol. 3005. Berlin, Germany: Springer; 2004. p. 11–20.
[15] Bui TN, Youssef WA. An enhanced genetic algorithm for DNA sequencing by hybridization with positive and negative errors. In: Deb K, Poli R, Banzhaf W, Beyer H-G, Burke EK, Darwen PJ, Dasgupta D, Floreano D, Foster JA, Harman M, Holland O, Lanzi PL, Spector L, Tettamanzi A, Thierens D, Tyrrell AM, editors. Proceedings of the GECCO 2004—genetic and evolutionary computation, Conference lecture notes in computer science, vol. 3103. Berlin, Germany: Springer; 2004. p. 11–20.
[16] Błażewicz J, Formanowicz P, Kasprzak M, Markiewicz WT, Weglarz J. Tabu search for DNA sequencing with false negatives and false positives. European Journal of Operational Research 2000;125:257–65.
[17] Błażewicz J, Glover F, Kasprzak M. DNA sequencing—Tabu and scatter search combined. INFORMS Journal on Computing 2004;16(3): 232–40.
[18] Fernandes ER, Ribeiro CC. Using an adaptive memory strategy to improve a multistart heuristic for sequencing by hybridization. In: Nikoletseas SE, editor. Proceedings of WEA 2005—4th international workshop on experimental and efficient algorithms, Lecture notes in computer science, vol. 3503. Berlin, Germany: Springer; 2005. p. 4–15.
[19] Blum C, Yábar Vallès M. New constructive heuristics for DNA sequencing by hybridization. In: Bücher P, Moret BME, editors. Proceedings of WABI 2006—6th International Workshop on Algorithms in Bioinformatics, Lecture notes in bioinformatics (LNBI), vol. 4175. Berlin, Germany: Springer; 2006. p. 355–65.
[20] Dorigo M, Stützle T. Ant colony optimization. Boston, MA: MIT Press; 2004.
[21] Blum C, Dorigo M. The hyper-cube framework for ant colony optimization. IEEE Transactions on Systems, Man, and Cybernetics—Part B 2004;34(2):1161–72.
[22] Walshaw C. Multilevel refinement for combinatorial optimization problems. Annals of Operations Research 2004;131.