Ann Oper Res (2013) 207:27–41
DOI 10.1007/s10479-011-0927-y

A hyper-heuristic approach to sequencing by hybridization of DNA sequences

Jacek Blazewicz · Edmund K. Burke · Graham Kendall · Wojciech Mruczkiewicz · Ceyda Oguz · Aleksandra Swiercz

Published online: 6 July 2011

Abstract In this paper we investigate the use of hyper-heuristic methodologies for predicting DNA sequences. In particular, we utilize Sequencing by Hybridization. We believe that this is the first time that hyper-heuristics have been investigated in this domain. A hyper-heuristic is provided with a set of low-level heuristics and the aim is to decide which heuristic to call at each decision point. We investigate three types of hyper-heuristics. Two of these (simulated annealing and tabu search) draw their inspiration from meta-heuristics. The choice function hyper-heuristic draws its inspiration from reinforcement learning. We utilize two independent sets of low-level heuristics. The first set is based on a previous tabu search method, with the second set being a significant extension to this basic set, including utilizing a different representation and introducing the definition of clusters. The datasets we use comprises two randomly generated datasets and also a publicly available biological dataset. In total, we carried out experiments using 70 different combinations of heuristics, using the three datasets mentioned above and investigating six different hyper-heuristic algorithms. Our results demonstrate the effectiveness of a hyper-heuristic approach to this problem domain. It is necessary to provide a good set of low-level heuristics, which are able to both intensify and diversify the search but this approach has demonstrated very encouraging results on this extremely difficult and important problem domain.

Keywords Hyper-heuristics · Simulated annealing · Tabu search · Choice function · Sequencing by hybridization

J. Blazewicz · W. Mruczkiewicz · A. Swiercz (✉)
Institute of Computing Science, Poznan University of Technology, Poznan, Poland
e-mail: aswiercz@cs.put.poznan.pl

J. Blazewicz · A. Swiercz
Institute of Bioorganic Chemistry, Polish Academy of Science, Poznan, Poland

E.K. Burke · G. Kendall
School of Computer Science, University of Nottingham, Jubilee Campus, Nottingham, UK

C. Oguz
Department of Industrial Engineering, Koç University, Sariyer, Istanbul, Turkey

1 Introduction

Biologically inspired problems are becoming more and more common in the field of computing science. Bioinformatics and computational biology issues have recently emerged at the intersection of the two disciplines which have illustrated a clear scientific synergy between molecular biology and operational research problems. A significant number of methods have been developed for analyzing the large amounts of experimental data and new tools have been created for predicting the structure of molecules. One of the key biochemical problems, which require engagement of computational methods is the reading of deoxyribonucleic acid (DNA) sequences. The reading of DNA sequences cannot be done at once, because genome sequences are too long. For example: a genome of bacteria is composed of $1–10 \times 10^6$ nucleotides, fruit fly—$1, 2 \times 10^8$, and human $3, 3 \times 10^9$. The sequencing instrumentation allows reading at once 50–600 nucleotide long sequences (depending on the method). These sequences must be merged together (with computational methods) in order to obtain the sequence of a whole genome. One of the methods for reading a DNA sequence on the low-level (600 nucleotide long) is called *sequencing by hybridization (SBH)* (Southern 1988). The method is based on a biochemical experiment in which the subfragments of the examined DNA sequence are determined. In order to reconstruct the sequence, the subfragments which overlap, are merged together. This method needs computational algorithms in the last phase of merging the fragments. Although the SBH problem has been addressed in the literature with significant success (Lysov et al. 1988; Dramanac et al. 1989; Pevzner 1989; Blazewicz et al. 2000, 2002a, 2002b, 2004, 2006; Bui and Youssef 2004; Blazewicz and Kasprzak 2003), this paper presents a unique insight from the perspective of developing fundamentally more generic search techniques in this area.

Hyper-heuristics, in the context of this paper, are search methodologies that manage a set of heuristics in order to solve the problem at hand. At each decision point, the hyper-heuristic decides which heuristic to execute from the set of those available (Burke et al. 2003a; Ross et al. 2003; Ross 2005). That is, a hyper-heuristic approach operates on the heuristic search space, rather than operating on a direct representation of the problem. The hyper-heuristic calls heuristic(s), which modifies the solution. The solution is next evaluated by the objective function. The heuristic(s) returns to the hyper-heuristic information about the improvement or deterioration of the objective value. Based on the behavior of the heuristic(s), the hyper-heuristic learns in which order the heuristics should be called. Figure 1 presents a schematic of the interaction between the hyper-heuristic, the low-level heuristics and solutions.

A key aim of hyper-heuristic approaches is to enable the search methodology to operate across different problem instances, or even different problem domains, without having to manually adapt the search algorithm. It may be necessary to supply a different set of heuristics but, for problems drawn from the same domain, this is often not necessary. In contrast, meta-heuristic approaches search over a direct representation of the problem and are usually developed as a bespoke algorithm for the problem being tackled (see Burke and Kendall 2005 for a comparison of hyper-heuristic and meta-heuristic approaches). Developing specialized algorithms for each problem domain (or even separate problem instances) is usually labor intensive and require parameter tuning to provide acceptable solutions. Hyper-heuristic approaches aim to adapt themselves to the problem at hand, and thus require less development time and less parameter tuning. Whilst hyper-heuristics may not return solutions which are as good as a bespoke algorithm, they are often able to return

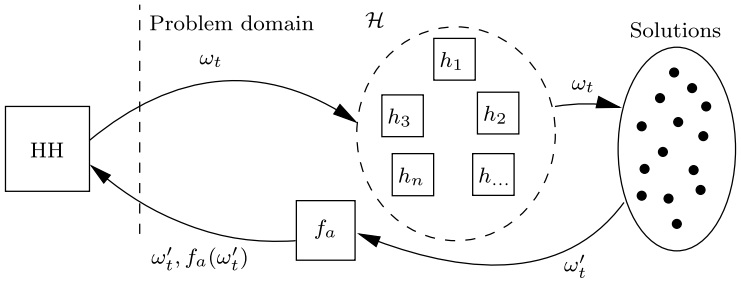

Fig. 1 Hyper-heuristic (HH) selects one (or more, depending on the hyper-heuristic mechanism) low-level heuristic(s) from set $\mathcal{H}$. The low-level heuristic(s) modifies the current solution, which is evaluated by the objective function f_a. The evaluation, together with new solution ω'_t, is sent back to the hyper-heuristic. If more than one heuristic was utilised, the hyper-heuristic chooses one of the solutions for the next iteration

solutions which are 'good enough, soon enough and cheap enough' (Burke et al. 2003a, 2003b).

In this paper, we investigate three different hyper-heuristic algorithms, simulated annealing (Kirkpatrick et al. 1983; Aarts et al. 2005), tabu search (Glover and Laguna 1997; Gendreau and Potvin 2005) and choice function. Simulated annealing based hyper-heuristics have been reported for the problem of container packaging in order to reduce waste (Dowsland et al. 2007), printed circuit board assembly (Ayob and Kendall 2003) and automating the design of planograms (supermarket shelf layouts) (Bai and Kendall 2005). Tabu search inspired hyper heuristics have been investigated across several domains including timetabling and rostering (Burke et al. 2003b; Burke and Soubeiga 2003) and examination timetabling (Burke et al. 2007; Kendall and Hussin 2005a, 2005b; Han and Kendall 2003). Choice function hyper-heuristics (Cowling et al. 2001) evaluates the past behavior of heuristics and uses this information to decide which heuristic to call next. This methodology has been utilized on a sales summit scheduling problem (Cowling et al. 2001), scheduling project presentations (Kendall et al. 2002) and nurse rostering problems (Cowling et al. 2002).

In this paper, the performance of different types of hyper-heuristics is analyzed. Our approach changes the set of moves (heuristics), which are called during the computations. The sensitivity of modifications is examined. In an ideal case, hyper-heuristic methods should work equally well, with every reasonable set of moves. The goal of our approach is to automatically learn and predict which moves might improve the solution. The aim of the paper is to establish a verifiable use case for hyper-heuristic algorithms based on the solution to the SBH problem. Next, the hyper-heuristics are tested on data coming from real DNA sequences and proved their efficiency. The hyper-heuristic framework can be used to solve other problems, by adapting the set of heuristics to the new problem in hand.

The paper is organized as follows. Next section describes the sequencing by hybridization problem in more details. Section 3 describes different hyper-heuristic algorithms which were tested in this study. In the following section low-level heuristics are presented. The section is divided into two parts. The first subsection presents a 'basic approach', where the set of low-level heuristics consists of simple moves. The more advanced approach is described in the second subsection. Section 5 presents the computational experiment and the last section contains final remarks.

2 Sequencing by hybridization problem (SBH)

Sequencing by hybridization is the process of reading a DNA sequence, that is establishing the order of DNA molecules, called *nucleotides* (Southern 1988). Nucleotides differ in nitrogenous bases, and can be distinguished into four groups: adenine, cytosine, guanine and thymine, abbreviated as A, C, G and T. The reading of a DNA sequence means giving the order of nucleotides, e.g. TAAGTCACG.... The SBH process includes two phases.

In the first phase, a biochemical experiment is performed in a solvent where the unknown, fluorescently labeled DNA sequence hybridizes to a DNA chip, which consists of an oligonucleotide library. An oligonucleotide library is the set of all possible DNA fragments (*oligonucleotides*) of a given length (usually 8–10 nucleotides). The examined DNA sequence hybridizes only to these oligonucleotides, which are complementary on the chip. As a result of the experiment, a spectrum is obtained, that is a set of oligonucleotides. These are the subfragments of the unknown DNA sequence being examined. On the fluorescent image of the chip, it is easy to see where the hybridization occurred and by knowing the coordinates of each oligonucleotide on the chip we can obtain the spectrum.

In the ideal experiment, the spectrum would consist of all subfragments of an unknown DNA sequence. In the real hybridization experiment, two types of errors can occur. Positive errors are represented by the oligonucleotides, which are in the spectrum but are not the subfragments of the examined sequence. Negative errors occur when the oligonucleotides, which are missing in the spectrum, but are the subfragments of the DNA sequence. Errors may come from erroneous reading of the chip image or from low quality image, and, in the case of negative errors, also from the repetitions of oligonucleotides in the sequence (oligonucleotides which appear in the sequence more than once, but are only once in the spectrum).

The second, computational phase of SBH is the reconstruction of an unknown DNA sequence from the elements in the spectrum. In the ideal experiment, the sequence is reconstructed from all the elements from the spectrum in such a way that the consecutive oligonucleotides are shifted by one position. The reconstruction of the DNA sequence is an easy problem (Pevzner 1989). The existence of any types of errors makes it impossible to construct the DNA sequence in polynomial time (Blazewicz and Kasprzak 2003).

Further on, only the real hybridization experiment (with errors) will be considered. The computational part of the SBH problem with positive and negative errors can be defined as follows (in the optimization version):

Parameters: Set S of oligonucleotides of equal length l, length n of an original sequence.

Answer: A sequence of length $\leq n$ containing the maximum number of elements from S.

An example of the computational part of the SBH problem is presented in Fig. 2.

3 The hyper-heuristic approach

The hyper-heuristics analyzed in this paper are derived from three different heuristic approaches—Choice Function, Tabu Search and Simulated Annealing. All of them fit in the framework that distinguishes two phases at every iteration of the search. The first phase

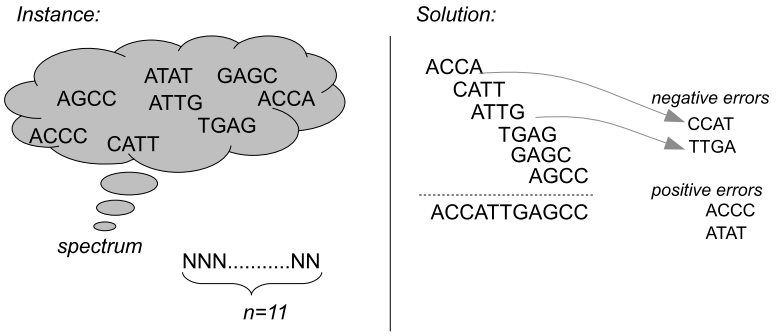

Fig. 2 An example of the SBH problem. As the input we obtain spectrum $S = \{\text{ACCA, ACCC, AGCC, ATAT, ATTG, CATT, GAGC, TGAG}\}$, which is an output of the hybridization experiment, and n—the length of the examined sequence. Overlapping oligonucleotides are merged together. The idea is to pack the most number of spectrum elements into the solution. In the places where oligonucleotides are shifted by more than 1 position, negative errors occur. If some elements from S are not used in the solution, they are positive errors. As an output we get the sequence created from the elements in the solution: ACCATTGAGCC

is a *selection* of low-level heuristics i.e., a single low-level heuristic is chosen from the set of available low-level heuristics. The second phase is called the *acceptance phase*. During this phase, it is decided whether the new solution obtained from the application of the selected low-level heuristic to the previous solution should be accepted or rejected.

```
HyperHeuristicSearch(H, ω_0)
1  t ← 0
2  while termination conditions are not satisfied
3      do h ← SelectHeuristic(H)
4         if AcceptHeuristic(h) = true
5            then ω_t ← h(ω_{t−1})
6            else ω_t ← ω_{t−1}
7         t ← t + 1
```

In the procedure HyperHeuristicSearch the set of low-level heuristics $\mathcal{H}$ and an initial solution ω_0 is given on input. During each turn the hyper-heuristic selects one low-level heuristic and uses it to obtain new ω_t when accepted. If h is rejected by the acceptance procedure then the current solution remains unchanged, $\omega_t = \omega_{t-1}$.

At the first phase (selection), the computation of a new solution can be very expensive and various techniques try to optimize the number of applied heuristics during each turn. The idea presented in Cowling et al. (2001) is the *choice function*, a hyper-heuristic, which uses three functions f_1, f_2 and f_3 to measure and rank low-level heuristics.

$$F(t,h) = \alpha f_1(t,h) + \beta f_2(t,h) + \gamma f_3(t,h) \tag{1}$$

Functions f_1 and f_2 are designed to intensify search. Function f_3 introduces an element of diversification. Function $f_1(t,h)$ for a low-level heuristic h, and time t, adds the ratio of the change of the value of the old solution to a new solution value obtained after applying h, and the amount of time spent for calling h the last time. Function $f_2(t,h)$ for a low-level heuristic h and its preceding heuristic g, is the ratio of the change in the solution when

applying the pair h directly after g and the time spent for calling f_2 the last time. The goal of function f_2 is to find and promote a co-operational behavior of two low-level heuristics. Function $f_3(t, h)$ is equal to the amount of time passed since h was used the last time. The choice function ranks all available low-level heuristics.

This rank is used to define four different selection methods. The straight choice (STRAIGHTC) algorithm selects the low-level heuristics with the greatest value of choice function.

```
STRAIGHTC(H)
1  h ← NULL
2  for l ∈ H
3       do if h = NULL or F(t, l) > F(t, h)
4            then h ← l
5  return h
```

Ranked choice (RANKEDC) evaluates k low-level heuristics with the greatest value of choice function F and selects the one which gives the greatest improvement on the solution value. At the start, set $\mathcal{H}'$ is equal to the set $\mathcal{H}$. In lines 3–7, k heuristics with the highest value of the choice function are checked (the output of *argmax* function). After every iteration the best performing heuristic is removed from set $\mathcal{H}'$. At the end, heuristic h with the highest value of the objective function f_a (this function measure the number of oligonucleotides from the spectrum in the solution) is chosen.

```
RANKEDC(H)
1  h ← NULL
2  H' ← H
3  for i = 1 to k
4       do g ← any heuristic from argmax_{g∈H'} F(t, g)
5          if h = NULL or f_a(g(ω_{t−1})) > f_a(h(ω_{t−1}))
6             then h ← g
7          H' ← H' \ g
8  return h
```

Decomp choice (DECOMPC) selection evaluates at most four low-level heuristics with the greatest values of f_1, f_2, f_3 and choice function F, and selects the low-level heuristics with the greatest improvement of f_a.

```
DECOMPC(H)
1  h ← any heuristic from argmax_{g∈H} F(t, g)
2  for i = 1 to 3
3       do g ← any heuristic from argmax_{g∈H} f_i(t, g)
4          if f_a(g(ω_{t−1})) > f_a(h(ω_{t−1}))
5             then h ← g
6  return h
```

Roulette choice (ROULETTEC) selects the low-level heuristic at random with probability function p.

$$p(h) = \frac{F(t, h)}{\sum_{g \in \mathcal{H}} F(t, g)} \tag{2}$$

ROULETTEC($\mathcal{H}$)
1 $h \leftarrow$ heuristic from set $\mathcal{H}$ chosen at random according to p
2 **return** h

All of those selections work with acceptance method, which accepts all moves. This gives four different hyper-heuristics.

ALLMOVES(h)
1 **return** TRUE

Tabu search selection (TABUS) is based on classical tabu search method. Tabu search algorithm as a hyper heuristic has already been used in Burke et al. (2003b), Kendall and Hussin (2005b). Here, the tabu search method works on low-level heuristics set. All low-level heuristics initially have a rank equal to zero. At every iteration, the low-level heuristic h with highest rank is selected and a new solution is obtained. If this solution brings any improvement to f_a, then the rank of h is increased by one. If the solution is not improved then the rank is decreased by one and h is put on the tabu list which prevents it from being used for k iterations. The rank is bounded between a fixed minimum and maximum. Tabu search selection together with the all moves acceptance forms the fifth hyper-heuristic. Pseudo code of tabu search hyper-heuristic is presented in Mruczkiewicz (2009).

The last hyper-heuristic is based on the simulated annealing meta-heuristic. It ranks low-level heuristics in exactly the same way as in tabu search selection but this rank is used as a probability of selecting low-level heuristic l. The hyper-heuristic SIMANN selects l at random with weighted probability specified by the current rank. The acceptance method, different than for the other hyper-heuristics, is called MONTECARLO. It accepts improved solutions always. If a solution is worsened then it is rejected with some probability, which is an exponential function of the solution objective value change divided by the current temperature. The current temperature $\delta(t)$ is given by

$$\delta(t) = \beta \log^{-1}\left(\left\lfloor \frac{t}{m} \right\rfloor + 2\right), \tag{3}$$

where t is the time passed from the beginning, and m is the number of iterations, when the temperature is constant during the cooling process.

MONTECARLO(h)
1 $\Delta \leftarrow f_a(h(\omega_{t-1})) - f_a(\omega_{t-1})$
2 **if** $\Delta > 0$
3 **then return** TRUE
4 **else if** $\exp\left(\frac{\Delta}{\delta(t)}\right) <$ random()
5 **then return** TRUE
6 **return** FALSE

Function random() returns a uniformly distributed random number in the range [0, 1). At the beginning, the temperature is high and the algorithm is more likely to accept deteriorated solutions. As time passes, the temperature cools down and the method is less likely to accept worse solutions. This temperature function is based on Bai et al. (2007) but it has a slightly different characteristic.

4 Low-level heuristics

Two independent sets of low-level heuristics are designed in order to solve the SBH problem. The first set is based on tabu search method described in Blazewicz et al. (2000). The second one is an extension of heuristics from the basic approach, which enables the choice of a more variable set of low-level heuristics.

4.1 Basic method

The moves described in Blazewicz et al. (2000) are used as a template for the first implementation of the low-level heuristics. In the mentioned tabu search algorithm, the solution is encoded using two collections. The first collection is *an ordered set of oligonucleotides* that occur in the final sequence. This ordered set is represented as a list and from this list, a DNA sequence is reconstructed. The second collection is *an unordered trash set*. Those two structures encode solutions that are available to this method. The sequence is reconstructed with the help of a greedy algorithm that traverses a list and tries to append every oligonucleotide at the closest position possible to the end of the constructed sequence. The reconstructed sequence might have length greater than n. Thus, feasible solutions are only those with length not greater than n.

The new objective function is constructed for the evaluation of the new solution. It is a linear combination of two solution measures—the number of used oligonucleotides and the solution density, which is a ratio of the solution length to the number of oligonucleotides used. The first measure is an objective value derived from the problem definition. The second part is required to introduce diversification and to prevent the search from being trapped in local maxima.

The moves described in Blazewicz et al. (2000) operate on single oligonucleotides and on clusters. The concept of the cluster of oligonucleotides is introduced as a sequence of following oligonucleotides that are shifted by only one nucleotide with the preceding oligonucleotide. Clusters are maximally condensed groups of oligonucleotides. They are very valuable because they give a large increase in the objective value and do not lengthen the sequence too much. Some of those moves are directly translated to the low-level heuristics. There are five different low-level heuristics.

- Insertion of an oligonucleotide. A single oligonucleotide from the trash set is inserted into the solution. There are no constraints on the destination position where the oligonucleotide can be put. The low-level heuristic tries to insert every unused oligonucleotide into every possible position and chooses the best option i.e., the combination with the greatest objective value.
- Shift of an oligonucleotide. An oligonucleotide is shifted from one position in the list to another. During the shift no cluster can be destroyed. Only oligonucleotides outside the cluster may be shifted provided that they do not break any cluster. Every possibility is checked.
- Shift of a cluster to another position in the list. A cluster can be shifted only if it does not break another cluster. Again, every possible combination of the shifted cluster and the destination position is checked.
- Deletion of the oligonucleotide. A single oligonucleotide from the solution is moved to the trash set. Only oligonucleotides outside the cluster or being at one of the cluster ends may be deleted. The oligonucleotide that gives the greatest objective value is used.
- Deletion of the cluster. Every oligonucleotide from the cluster will be placed in the trash set. This low-level heuristic is quite invasive. It can break the solution and change it extensively.

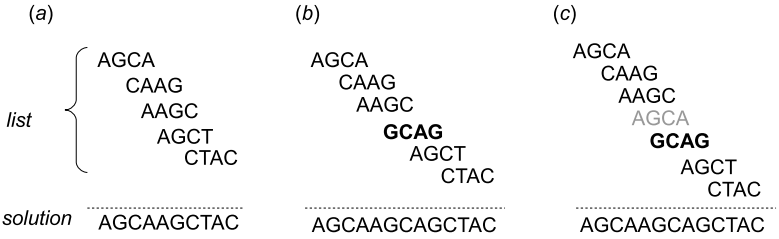

Fig. 3 Example of the solution in the extended method

The tabu search method described in Blazewicz et al. (2000) remembers inserted or shifted oligonucleotides in the tabu list. The oligonucleotides in the tabu list cannot be removed for a number of iterations. The purpose of those steps is to prevent the reversal of the moves made. The methodology of hyper-heuristic search precludes from copying the behavior of the tabu list. No substitution of the tabu list is introduced to the low-level heuristics implemented.

This set of low-level heuristics was used to solve instances of the SBH problem. The results obtained were satisfactory. However, in order to improve the solutions obtained a number of modifications have been introduced.

4.2 An extended method

The changes introduced to the basic approach are quite extensive. The solution encoding is different, the new definition of the clusters, and a new low-level heuristic are introduced. The solution is no longer encoded as a permutation of oligonucleotides. The extended approach uses a list, which allows repetitions of oligonucleotides on the list. An example of the representation of the solution, and encoding it into a sequence is presented in Fig. 3. Spectrum S is composed of the following oligonucleotides $S = \{$AAGC, AGCA, AGCT, CAAG, CTAC, GCAG$\}$. The solution is composed of the five elements of the spectrum, as presented in Fig. 3(a). Inserting an oligonucleotide GCAG into the solution (Fig. 3(b)) will indirectly induce inserting also an oligonucleotide AGCA (Fig. 3(c)). Thus, oligonucleotide AGCA appears in the list twice. The list that encodes the solution guarantees that every oligonucleotide present in the solution is present on the list. Inserting the new oligonucleotide into the list always changes the solution.

Apart from the heuristics from the basic method (insertion, deletion, and shift) a new low-level heuristic—swap—is introduced. It is a combination of oligonucleotide deletion and insertion. Every low-level heuristic is also given in a few variants: the oligonucleotide, which is going to be modified (inserted, shifted, etc.) can be chosen randomly or each one is checked. Also, the position on the list can be chosen at random or determined by checking every possible position. This leads to four different insert, swap and shift heuristics and two different delete heuristics (the destination cannot be chosen, because it is always an unordered trash set).

All the new low-level heuristics are now operating on clusters instead of single oligonucleotides, but the definition of the cluster differs from the basic approach. The cluster of oligonucleotides is extended to include gaps between oligonucleotides in cluster. The number of consecutive missing oligonucleotides is now a parameter that defines the type of a cluster. The number can vary from 1 to l (l is the length of oligonucleotides). If a cluster has a parameter 1, then consecutive oligonucleotides are shifted by one position. If a cluster parameter is l, then the cluster consists of the whole solution (the oligonucleotides in

the solution are maximally overlapped, so they can be shifted by at most l positions). This new definition of clusters allows parameterizing all of the low-level heuristics by the type of a cluster that they can operate on. Those parameters lead to a large variety of low-level heuristics with dramatically different behavior. They were compared and an influence on the search process was measured.

5 Computational experiment

The computational experiments were performed in two phases. In the first phase, the behavior of different hyper-heuristics was investigated, as well as the sensitivity to the different sets of low-level heuristics. In order to check all possible combination of heuristic sets and parameter values, the test bed was performed for the randomly generated data. Over 70 different combinations of low-level heuristic sets were tested against six hyper-heuristics. Next, the biological dataset was tested only for the best configurations of heuristic sets and hyper-heuristics. The results on the random dataset and on the biological dataset are presented in Sect. 5.1 and in Sect. 5.2, respectively.

All the tests were conducted on IBM PC with Intel Core2 Duo 3.16 GHz processor and 2048 MB of physical memory, running WindowsXP operating system. The algorithms were implemented in Java, and compiled and run in the Java runtime environment: Sun's JRE 6 Update 13.

5.1 Random dataset

Two random generators were implemented in order to generate new instances. The first generated instances which simulate a hybridization experiment for a randomly generated sequence. The generator also adds some positive and negative errors. We also generated random instances, which abstract from the biological nature of the problem and which generates totally random instances without any correlation between oligonucleotides.

The first generator was very helpful in comparing exact methods and testing the correctness of the hyper-heuristic methods. The second one generated instances for the test bed. Of all generated instances, only the most diverse (in the sense of solution domain) 20 were selected. The size of the spectrum was in the range 50–4000 oligonucleotides, the length of each oligonucleotide changed from 4 to 12. The length of the examined sequence was in the range 50–800.

The results of six hyper-heuristics with different low-level heuristic sets for the basic approach (Sect. 4.1) are presented in Fig. 4. Configuration 'a' is the set of all possible low-level heuristics except 'delete cluster' heuristic, while configuration 'b' is composed of all low level heuristics.

The comparison of various hyper-heuristics, in the case of the basic approach shows that two hyper-heuristics perform noticeably differently from the others. In the case of configuration 'a', i.e. a good selection of low-level heuristics, where the 'delete cluster' is not present, simulated annealing (SIMANN) produces the best solutions. This is caused by its random factor (the Monte Carlo acceptance procedure) which by occassionally worsening the solution achieves better global result. In case of bad choice of low-level heuristics (configuration 'b'—all heuristics) simulated annealing is not powerful enough to choose the best set of moves; the constructed solutions are below average. In both cases ROULETTEC produces solutions which are very poor. Other hyper-heuristics quickly learn the set of profitable moves (which low-level heuristics are to be used).

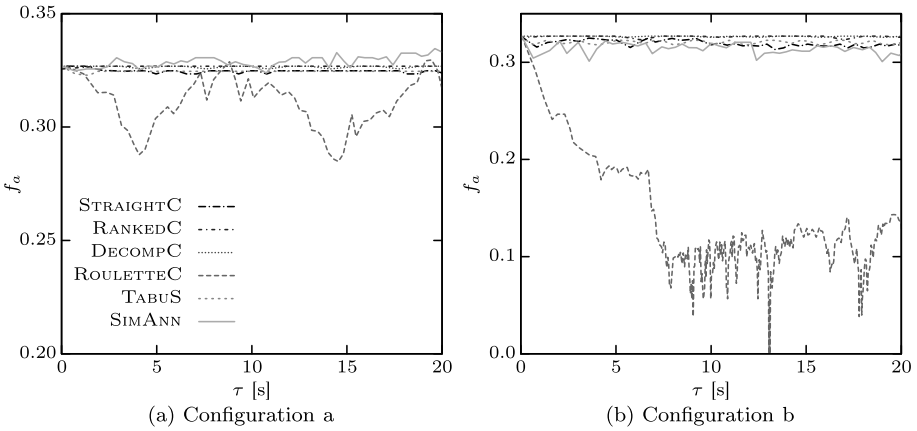

Fig. 4 Basic approach. Results for different low-level heuristics configurations. On the X axis there is time of computations, and on the Y axis—the objective function value

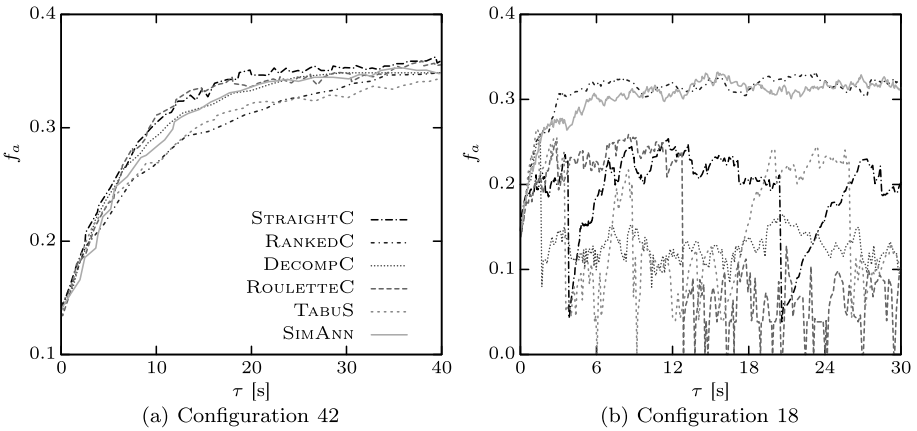

Fig. 5 Extended approach. Results for different low-level heuristics configurations

For the extended approach, 70 different configurations were designed. Each configuration of the low-level heuristics set differs in the types of heuristics (insert, shift, swap and delete) used, in the randomness (or the lack of it), and also in the cluster parameters. Figure 5 presents the results obtained for two configurations which resulted in the most striking results. The results obtained by configuration no. 42, which was composed of 10 different heuristics, were the best among all other configurations. In Fig. 5(a), it can be seen that all of the hyper-heuristics performed in a comparable manner. Even the most randomly behaving hyper-heuristic RouletteC, performed similarly to the best. On the other hand, if the set of low-level heuristics is not properly configured, e.g. it is composed only of primitive or random heuristics, then the hyper-heuristics might not be able to learn the sequence, in which it has to call heuristics. The results for the badly configured heuristics set are shown in Fig. 5(b). The two hyper-heuristics, which were able to learn how to use heuristics, were RankedC and SimAnn. The same relation is also visible in Fig. 4: for configuration 'a'

all the hyper-heuristics could find similar solutions and performed reasonably well, while for configuration 'b', the RouletteC could not obtain good results.

5.2 Biological dataset

In the second set of tests the best performing configurations from the above experiment (random dataset) were used and tested on biological data. The biological datasets are publicly available at http://bioserver.cs.put.poznan.pl. These instances were used to test the methods in Blazewicz et al. (2006). The 40 spectra were derived from the DNA sequences published in GenBank (http://www.ncbi.nlm.nih.gov/genbank/), and coding human proteins. The accession numbers of the sequences can be found on the web site. The sequences were of length 200–600 nucleotides. Spectra of these sequences were created and experimental (positive and negative) errors were artificially introduced. Errors are added to the spectra by removing some of the oligonucleotides from the spectra (negative errors) and by adding some new random oligonucleotides to the spectra (positive errors). The number of removed or added oligonucleotides is determined by the error percentage. '5% of errors' means that 5% of oligonucleotides from the spectrum were removed and the same number of random oligonucleotides were added to the spectrum.

Spectra can be divided into two categories. In the first category the spectra are created from the sequences that do not contain any repetitions of oligonucleotides. The spectra contain 5% or 20% of experimental errors. The sequences containing repetitions of oligonucleotides were used to create the spectra in the second category. Here, the spectra contain 0% or 5% of errors. 0% of errors means that there are only negative errors coming from repetitions and the number of errors differ for each sequence.

The results of the computational experiments of different algorithms for the instances in the first category are shown in Table 1, and for the second category—in Table 2, respectively. Three measures evaluate each result. 'Avg usage' is the percentage of oligonucleotides from the spectrum used to construct the solution. Due to positive errors, 100% means that the solution was created from as many oligonucleotides as is the cardinality of spectrum minus the number of positive errors. 'Optimal count' is the number of instances out of 40, which were solved with 100% of the usage. The above measure can evaluate the solution from the mathematical point of view, but for the biologists the most important is the similarity of the solution to the examined sequence. The last measure calculates the number of the same letters in two sequences with the Needelman-Wunsch algorithm (Needleman and Wunsch 1970). 100% means here that the sequences are identical.

It has to be stressed here, that the only measure which can mathematically evaluate the solution is the number of oligonucleotides in the solution (the objective function). The usage and the similarity are not equivalent. Even for the ideal experiment it is possible to obtain different sequences from the same number of oligonucleotides. With the increase of errors the probability that for very high usage the alignment is not satisfactory also increases. Such situations can be seen in case of repetitions, in Table 2 when despite the usage is above 99%, the similarity is equal to 72–78%. In case of genetic algorithm for 100% of usage, the similarity is only 91–92%.

Four different algorithms that are presented in both tables, were chosen as the best ones in different categories. The first method is the best performing hyper-heuristic for the 'basic approach', SimAnn with configuration 'a' of low-level heuristics. For the extended approach RouletteC with configuration no. 42 surprisingly obtained the highest values of the objective function. Other algorithms presented in tables are two metaheuristics specialized for the SBH problem: hybrid genetic algorithm (Blazewicz et al. 2006) and tabu and scatter search (Blazewicz et al. 2004).

Table 1 The results of the tests of different algorithms for biological dataset. Spectra are created from DNA sequences which do not contain repetitions of oligonucleotides. In the first row there is the length of the examined sequence, next one is the percentage of errors in the spectrum. Each entry in rows 'Avg. usage' and 'Alignment' is the mean value out of 40 instances

Instance	200		400		600	
	5%	20%	5%	20%	5%	20%
Hyper-heuristic SIMANN + configuration 'a'—basic approach						
Optimal count	37/40	34/40	35/40	22/40	30/40	14/40
Avg. usage [%]	99.81	99.64	99.83	98.93	99.66	98.20
Alignment [%]	98.06	91.48	95.69	82.00	93.25	74.18
Hyper-heuristic ROULETTEC + configuration no. 42—extended approach						
Optimal count	40/40	40/40	36/40	37/40	23/40	5/40
Avg. usage [%]	100.00	100.00	99.89	99.92	98.94	97.20
Alignment [%]	99.52	98.74	95.61	94.68	92.92	86.03
Hybrid genetic algorithm (Blazewicz et al. 2006)						
Optimal count	40/40	40/40	40/40	40/40	40/40	40/40
Avg. usage [%]	100.00	100.00	100.00	100.00	100.00	99.98
Alignment [%]	100.00	99.84	100.00	100.00	99.87	99.81
Tabu and scatter search (Blazewicz et al. 2004)						
Optimal count	39/40	39/40	37/40	28/40	32/40	20/40
Avg. usage [%]	99.93	99.93	99.90	99.67	99.84	99.36
Alignment [%]	99.87	98.44	98.96	95.70	95.82	88.50

Analyzing the results in the tables it can be observed that hyper-heuristics results in the solutions of high usage, similar to the metaheuristics, which were specialized for the SBH problem. SIMANN appeared to be the best for the basic approach, what proved also the results obtained for the random dataset. Surprisingly, the ROULETTEC found the best solutions. Its behavior was analyzed also in Sect. 5.1, where roulette choice function destroyed the solutions. Here, for a good set of low-level heuristics all the hyper-heuristics were searching around local optimum, while ROULETTEC, by destroying a good solution, could jump into different place in the solution space and resulted in finding better solutions.

Thus, it is important to design a good set of low-level heuristics: some heuristics which aim to intensify the search and some random which can destroy solutions, and diversify the search.

6 Final remarks

In this paper we have investigated the use of hyper-heuristics for predicting DNA sequences. This is an extremely challenging domain and many bespoke algorithms struggle to find good quality solutions. We have demonstrated that hyper-heuristics are an effective approach, which is worthy of further investigation. In particular, it is important to provide a set heuristics that provides sufficient coverage of the search space. It would be interesting to find out what is the minimum set of heuristics that is required in this domain. Of course, we can provide more heuristics than are required (to ensure coverage) but this means that the hyper-

Table 2 The results of computational tests of different algorithms for the dataset of second category, where the instances contain negative errors coming from repetitions (0%—errors coming from repetitions only; 5%—additionally 5% of positive and negative errors are added). All the sequences were of length 600

Instance	Optimal Count	Avg usage [%]	Alignment [%]
Hyper-heuristic SIMANN + configuration 'a'—basic approach			
Rep 0%	17/40	99.4	74.0
Rep 5%	20/40	99.2	73.0
Hyper-heuristic ROULETTEC + configuration no. 42—extended approach			
Rep 0%	14/40	99.2	78.6
Rep 5%	15/40	99.3	77.4
Hybrid genetic algorithm (Blazewicz et al. 2006)			
Rep 0%	40/40	100.0	92.7
Rep 5%	40/40	100.0	91.2
Tabu and scatter search (Blazewicz et al. 2004)			
Rep 0%	33/40	99.8	88.4
Rep 5%	23/40	99.5	82.6

heuristic has to learn not to use some of the heuristics which, of course, adds to the computational time. Of the hyper-heuristics investigated, both SIMANN and ROULETTEC were the most promising, producing the best quality solutions of all those investigated. There is significant scope to explore other hyper-heuristics and, given the promising results reported in this paper, there is significant research potential in doing this.

Acknowledgements The work has been partially supported by the NCN grant.

References

Aarts, E., Korst, J., & Michiels, W. (2005). Simulated annealing. In E. K. Burke & G. Kendall (Eds.), *Search methodologies: introductory tutorials in optimization and decision support techniques* (pp. 187–210). Berlin: Springer. Chap. 7.

Ayob, M., & Kendall, G. (2003). A Monte Carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. In *Placement machine*, InTech'03 Thailand (pp. 132–141).

Bai, R., & Kendall, G. (2005). An investigation of automated planograms using a simulated annealing based hyper-heuristics. In T. Ibaraki, K. Nonobe, & M. Yagiura (Eds.), *Metaheuristics: progress as real problem solvers operations research/computer science interfaces series* (Vol. 32, pp. 87–108). Berlin: Springer.

Bai, R., Blazewicz, J., Burke, E. K., Kendall, G., & Mccollum, B. (2007). *A simulated annealing hyper-heuristic methodology for flexible decision support* (Tech. rep.). School of CSiT. University of Nottingham, UK.

Blazewicz, J., & Kasprzak, M. (2003). Complexity of DNA sequencing by hybridization. *Theoretical Computer Science*, *290*(3), 1459–1473.

Blazewicz, J., Formanowicz, P., Kasprzak, M., Markiewicz, W., & Weglarz, J. (2000). Tabu search for DNA sequencing with false negative and false positives. *European Journal of Operational Research*, *125*, 257–265.

Blazewicz, J., Formanowicz, P., Guinand, F., & Kasprzak, M. (2002a). A heuristic managing errors for DNA sequencing. *Bioinformatics*, *18*, 652–660.

Blazewicz, J., Kasprzak, M., & Kuroczycki, W. (2002b). Hybrid genetic algorithm for DNA sequencing with errors. *Journal of Heuristics*, *8*, 495–502.

Blazewicz, J., Glover, F., & Kasprzak, M. (2004). DNA sequencing—tabu and scatter search combined. *INFORMS Journal on Computing*, *16*, 232–240.

Blazewicz, J., Glover, F., Swiercz, A., Kasprzak, M., Markiewicz, W., Oguz, C., & Rebholz-Schuhmann, D. (2006). Dealing with repetitions in sequencing by hybridization. *Computational Biology and Chemistry*, *30*(5), 313–320.

Bui, T., & Youssef, W. (2004). An enhanced genetic algorithm for DNA sequencing by hybridization with positive and negative errors. *Lecture Notes in Computer Science*, *3103*, 908–919.

Burke, E. K., & Kendall, G. (Eds.) (2005). *Search methodologies: introductory tutorials in optimization and decision support techniques*. Berlin: Springer.

Burke, E. K., & Soubeiga, E. (2003). Scheduling nurses using a tabu-search hyperheuristic. In *Proceedings of the 1st multidisciplinary international conference on scheduling: theory and applications (MISTA 2003)*, 197–218.

Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., & Schulenburg, S. (2003a) Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of metaheuristics*. Dordrecht: Kluwer Academic. Chap. 16.

Burke, E. K., Kendall, G., & Soubeiga, E. (2003b). A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, *9*(6), 451–470.

Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., & Qu, R. (2007). A graph-based hyper-heuristic for timetabling problems. *European Journal of Operational Research*, *176*(1), 177–192.

Cowling, P., Kendall, G., & Soubeiga, E. (2001). A hyperheuristic approach to scheduling a sales summit. In *PATAT '00: Selected papers from the third international conference on practice and theory of automated timetabling III* (pp. 176–190). London: Springer.

Cowling, P., Kendall, G., & Soubeiga, E. (2002). Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In: *Lecture notes in computer science*, *EvoWorkShops*, pp. 1–10. Berlin: Springer.

Dowsland, K., Soubeiga, E. & Burke, E.K (2007). A simulated annealing hyper-heuristic for determining shipper sizes. *European Journal of Operational Research 179*(3), 759–774.

Dramanac, R., Labat, I., Brukner, I., & Crkvenjakov, R. (1989). Sequencing of megabase plus DNA by hybridization: Theory of the method. *Genomics*, *4*(2), 114–128.

Gendreau, M., & Potvin, J. Y. (2005). Tabu search. In: E. K. Burke & G. Kendall (Eds.), *Search methodologies: introductory tutorials in optimization and decision support techniques* (pp. 165–186). Berlin: Springer. Chap. 6.

Glover, F., & Laguna, M. (1997). *Tabu search*. Boston: Kluwer Academic.

Han, L., & Kendall, G. (2003). Investigation of a tabu assisted hyper-heuristic genetic algorithm. In *Proceedings of congress on evolutionary computation (CEC2003)* (Vol. 3, pp. 2230–2237).

Kendall, G., & Hussin, M. (2005a). A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA University of Technology. *Lectures Notes in Computer Science*, *3616*, 270–293.

Kendall, G., & Hussin, N. M. (2005b). In G. Kendall, E. Burke, S. Petrovic, & M. Gendreau (Eds.), *An investigation of a tabu-search-based hyper-heuristic for examination timetabling, multidisciplinary scheduling; theory and applications* (pp. 309–328). Berlin: Springer.

Kendall, G., Soubeiga, E., & Cowling, P. (2002). Choice function and random hyperheuristics. In *Proceedings of the 4th Asia-Pacific conference on simulated evolution and learning (SEAL'02)* (pp. 667–671).

Kirkpatrick, S., Gelatt, C.D., & Vecchi, M.P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

Lysov, I.P., Florent'ev, V.L., Khorlin, A.A., Khrapko, K.R., & Shik, V.V. (1988). Determination of the nucleotide sequence of DNA using hybridization with oligonucleotides. A new method. *Doklady Akademii Nauk SSSR*, *303*, 1508–1511.

Mruczkiewicz, W. (2009). *Hyper-heuristics for sequencing by hybridisation problem*. Master Thesis, Poznan University of Technology, Poland.

Needleman, S. B., Wunsch, C.D. (1970). A general method applicable to the search for similarities of the amino acid sequence of two proteins. *Journal of Molecular Biology 48*, 443–453.

Pevzner, P. A. (1989). 1-tuple DNA sequencing: computer analysis. *Journal of Biomolecular Structure and Dynamics*, *7*, 63–73.

Ross, P. (2005). Hyper-heuristics. In E. K. Burke & G. Kendall (Eds.), *Search methodologies: introductory tutorials in optimization and decision support techniques* (pp. 529–556). Berlin: Springer. Chap. 17.

Ross, P., Marin-Blázquez, J. G., Schulenburg, S., & Hart, E. (2003). Learning a procedure that can solve hard bin-packing problems: A new GA-based approach to hyper-heuristics. In *Proceedings of the genetic and evolutionary computation conference* (pp. 1295–1306). Berlin: Springer.

Southern, E. (1988). United Kingdom Patent Application GB8810400.

Zhang, J. H., LY, Wu, & Zhang, X. S. (2003). Reconstruction of DNA sequencing by hybridization. *Bioinformatics*, *19*(1), 14–21.