Hyper-Heuristics and Metaheuristics for Selected Bio-Inspired Combinatorial Optimization Problems

Aleksandra Swiercz

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.69225

Abstract

Many decision and optimization problems arising in bioinformatics field are time demanding, and several algorithms are designed to solve these problems or to improve their current best solution approach. Modeling and implementing a new heuristic algorithm may be time-consuming but has strong motivations: on the one hand, even a small improvement of the new solution may be worth the long time spent on the construction of a new method; on the other hand, there are problems for which good-enough solutions are acceptable which could be achieved at a much lower computational cost. In the first case, specially designed heuristics or metaheuristics are needed, while the latter hyper-heuristics can be proposed. The paper will describe both approaches in different domain problems.

Keywords: hyper-heuristics, bioinformatics

1. Introduction

Many heuristics and metaheuristics (problem-independent algorithmic framework) have been successfully applied for decision and optimization problems. However, there are difficulties in using the already-existing algorithms for new problems or even for new instances of a similar problem. Typically, one needs a time-consuming phase for parameters tuning. The parameters, are often not well-described and do not allow for solving the problem at a satisfactory level (producing satisfactory solutions). Thus, in many cases, one needs to construct new algorithms to solve particular instances. Recently, there have been attempts to automate the process of designing methods to learn the tuning of the parameters. The basic idea is to develop a method which is general and operates on small moves and learns which moves should be applied at each stage of the solving process.

The term hyper-heuristics was first used by Cowling et al. in 2001 [1] for the sales summit problem and refers to a method which does not use the problem-specific information other than a set of simple knowledge poor heuristics which are easy to implement. Between the set of simple low-level heuristics and high-level heuristics, there exists a domain barrier, which does not allow to pass the information about the problem domain. A high-level hyper-heuristic uses performance indicators when a low-level heuristics is called (indicators are not specific to the problem) in order to decide which heuristics should be chosen at a particular time point in the search space. However, the ideas behind hyper-heuristic are not new to the scientific community, and their roots trace back to 1963 and spanned several areas: computer science, artificial intelligence, and operational research. In 1963, Fisher and Thompson [2] showed that combining scheduling rules, known as dispatching or priority rules, is better when they are combined rather than used separately. In 1990s, the ideas were further explored. Storer et al. [3] designed a few fast problem-specific heuristics and defined neighborhoods within the search space. The approach could be applied with any scheduling objective and was tested for the job shop scheduling problems with the minimum makespan objective. In Fang et al. [4], a genetic algorithm (GA) was developed which tried to avoid difficulties in representing a solution as a chromosome, which is needed by the GA. The proposed method searches abstract regions of the solution space and then uses another method which converts the points generated by the GA into candidate solutions. Drechsler and Becker [5] presented a GA which first, based on benchmark examples, learns a good sequence of basic optimization modules (simple methods) and then applies it to instances of a given problem (computer-aided design of integrated circuits).

Some examples of automated parameter tuning can also be considered as a basis for hyper-heuristics. In Ref. [6], one evolutionary algorithm was used to tune the second one, which solved a particular problem. Also, some approaches of self-adaptation in the parameter tuning of the evolutionary algorithms were summarized in the survey [7].

In the area of machine learning, the idea of choosing the best algorithm for a given problem was first posed by Rice [8]. Following this idea, several projects created specialized systems to help to select/recommend the best method, as for example, Consultant-2 [9] and Teacher (Techniques for the Automated Creation of Heuristics) [10]. Consultant-2 was developed to support the machine learning toolbox. It integrates the knowledge of choosing the proper machine learning algorithms based on the nature of domain data and the domain experts' knowledge of preprocessing and manipulating the data, in order to use the system without directly involving the specialists. On the other hand, Teacher was designed as a system for learning and generalizing heuristics used in problem-solving. Despite the lack of or little domain knowledge, the system was able to improve the existing heuristic methods. The Teacher was successfully applied in the area of process mapping, load balancing, routing, and testing.

The above examples show that the term hyper-heuristics, although did not appear before the year 2000, had circulated in the literature for quite a long time. Since then, the term was used, and the idea further developed in many different problem domains [11, 12].

The methods often used in the context of hyper-heuristics have strong connections with biology. However, the flow of ideas between biology and operational research works in both directions. Observations of nature provide the basis for designing algorithms: many evolutionary and genetic algorithms were used as (hyper-)heuristics to solve combinatorial optimization problems. On the other hand, operational research methods can help solving problems arising in the field of biology and resulted in creating a new area of Bioinformatics. Some examples of problems solved with the use of bioinformatics tools are DNA sequencing, DNA mapping, RNA structure prediction and protein folding. Moreover, mutual infiltration of the ideas of the two scientific domains can work in both directions just for a single problem. The DNA sequencing by hybridization (SBH) problem (further described in Section 4) in the ideal case was first defined as searching for a Hamiltonian path in a special graph [13], which is an NP-hard problem. SBH was later modeled as the search for a Eulerian path that could be solved in polynomial time [14], which is the opposite of searching for a Hamiltonian path problem. Analysis of this phenomenon resulted in defining a new type of DNA graphs and showed why searching for a Hamiltonian and Eulerian path in these two types of graphs is equivalent [15]. For that particular problem, the operational research methods were used to solve the problem, while the analysis of SBH problem introduced a new type of graphs and gave an insight into the connections between easy and NP-hard problems.

The goal of this chapter is to show the connections between the areas of biology, computer science and operational research in the context of (hyper-)heuristics search. Although some surveys on the meta and hyper-heuristic search have been published [11, 12], here we focus mainly on the selected bio-inspired problems solved with hyper-heuristic methods. In the next section, the classification of hyper-heuristic algorithms is presented. Section 3 describes different methods used in the hyper-heuristic framework. The following section focuses on few biological problems successfully solved with hyper-heuristics. In Section 5, we show that not all the problems could be solved with hyper-heuristics and specially tailored-to-measure heuristics are needed. We also propose a small hint how hyper-heuristic search could be incorporated in solving the DNA assembly problem. Section 6 summarizes and highlights a potentially interesting future research direction.

2. Hyper-heuristics and their classification

A hyper-heuristic framework consists of a set of low-level heuristics and a high-level hyper-heuristic algorithm. The latter evaluates the performance of low-level heuristics and selects one of them to change the current solution. The performance can be measured as an increase in the objective function value, defined for the problem, but can also check the time of computations or the time a heuristic was last used. The hyper-heuristic can process one (single point search) or multiple solutions at a time (multi-point search). In the former, an initial candidate solution goes through a set of successive steps until it gets to the final solution. In the latter, utilized for the perturbative methods, a few solutions are processed in parallel, like for example in the AMALGAM approach, which operates on the population of solutions [16]. Apart from the selection of the low-level heuristics, the acceptance mechanism seems to be crucial in the hyper-heuristics research. The decision whether to accept or reject the new solution can be always the same for the same (*deterministic*) or different (*non-deterministic*) input, for example, dependent on the time passed. The process is repeated iteratively, a low-level heuristic is selected from the available ones in the set, the decision is made about the acceptance of the heuristic and in the case of acceptance, it is applied to the solution until a stopping criterion is met. The high-level heuristic has no information about the solved problem and is operating only on the heuristic search space, opposite to metaheuristics which operate on the solution space. The general scheme of the hyper-heuristic framework is presented in **Figure 1**.

In Burke et al. [17], the definition of a hyper-heuristic was further extended to *a search method or selection mechanism for selecting or generating heuristics to solve computational search problems*. The classification of hyper-heuristics presented in surveys [11, 12, 17] takes into account the nature of the heuristic search space and the feedback used for learning mechanism (see **Figure 2**).

According to the nature of the search space, we might have methodologies that *select* existing heuristics to use and methodologies that *generate* new heuristics on the basis of existing smaller components. The second level in this dimension corresponds to the *constructive* and *perturbative* methods. Perturbative methods are using complete candidate solutions and change them by modifying solution components, while constructive methods start from partial candidate solutions and extend them iteratively. This type of approach has been applied to several hard combinatorial problems such as educational timetabling [18–20], production scheduling [21], packing [22] or vehicle routing [23]. In the case of generation methods, hyper-heuristics search the space of heuristics constructed from components rather than well-defined heuristics. The examples where generation hyper-heuristics were used include several domains: timetabling and scheduling [24], the traveling salesman problem [25, 26] or cutting and packing [27–29]. Both classes of hyper-heuristics, selection, and generation, output a solution at the end of a run, but a heuristic generator outputs also new heuristics that produced the solution, and these heuristics could be potentially used for the next problem.

The second dimension corresponds to a learning mechanism which is used by a hyper-heuristic algorithm. If the learning takes place while the algorithm is solving an instance of the problem than we say that there is an *online* feedback. The idea is to learn a good sequence of heuristics for the problems at hand [1, 22, 30].

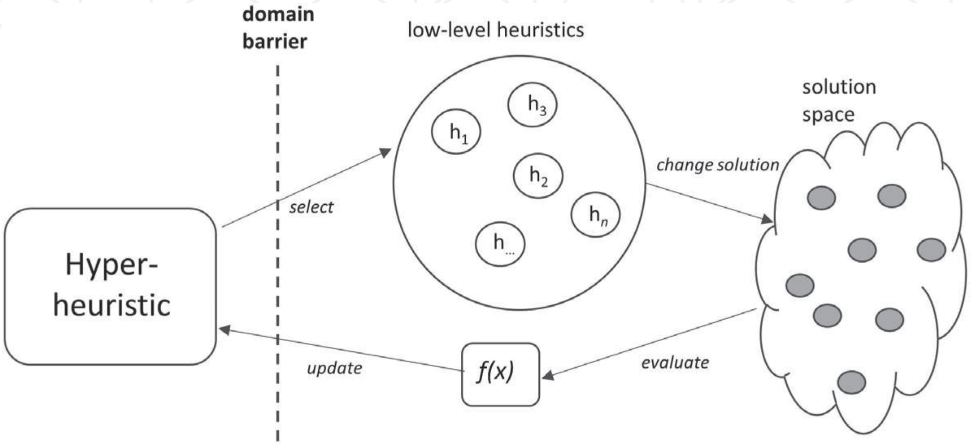

Figure 1. General scheme of how hyper-heuristics work.

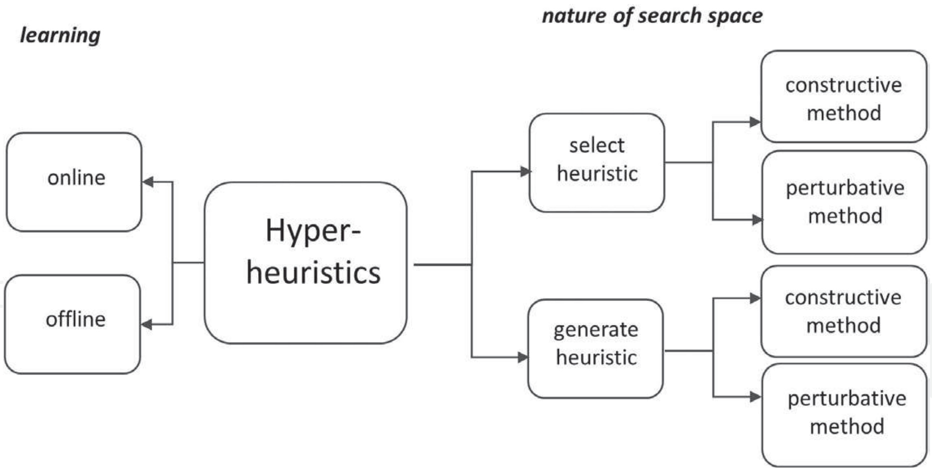

Figure 2. Classification of hyper-heuristics (following Ref. [11]).

In *offline* learning, the idea is to gather knowledge in the form of rules or programs while solving some benchmark instances and hope that these rules are general enough to solve unseen instances [31–33].

There are some methods which do not fit strictly to the frames presented above but rather span across two or more categories. They use either selecting and generating methods [34] or perturbative and constructive heuristics [35].

3. Learning techniques in hyper-heuristics

Learning techniques are key/essential indicators of the quality of hyper-heuristics. Good learning mechanisms impinge on obtaining better solutions and the possibility to reuse the hyper-heuristic and this decrease software production costs. Low-level heuristics are usually simple heuristics designed for a problem, but if used in a wrong order may not allow to get better solutions than when a single such heuristic is applied. Depending on the nature of the search space, different learning mechanisms were used. Below a few of the approaches are mentioned, some of them derived from the observations on biological evolution made by naturalist Charles Darwin. In his concept, the individuals in the population of one species are subject to the natural selection rules to fit the environment better. Among the fitting mechanisms, we can distinguish (i) crossover, reproducing of individuals in order to create a new one(s), (ii) mutation, random change of an individual in order to introduce diversity to the population, and (iii) selection of the best features in the population or in one individual. Biological evolution was an inspiration for developing bio-inspired algorithms like evolutionary algorithms, genetic algorithms or genetic programming. All of them were utilized as hyper-heuristics in different contexts of the nature of the search space. A few examples are mentioned in the following paragraphs.

In the selection mechanism with the combination of constructive heuristics, commonly used local search–based hyper-heuristics explore the solution space with the selected heuristics as widely as possible. A variable neighborhood search was used in the context of examination timetabling [18]. Tabu search-based hyper-heuristic was applied for the workforce scheduling problem [36] and course and exam timetabling [19]. Evolutionary or genetic algorithms were used for solving the vehicle routing problem [35] and bin packing of 2D elements [37].

In case of perturbative low-level heuristics, more commonly used are score-based hyper-heuristics. A choice function was introduced in Ref. [1] which evaluates each heuristic according to a score composed of three components: how well a heuristic performs, how well it performs when combined with another one, and the time elapsed since it was last used. A low-level heuristic can be selected in four different ways (i) *randomly*, (ii) taking the one giving the best value of the choice function (*greedy*), (iii) basing the choice on a *ranking* of best heuristics, or (iv) selecting a heuristic with the probability equal to the proportion of choice function value (*roulette wheel*). Choice function hyper-heuristics solved the sales summit problem [1], timetabling and scheduling [38] and sequencing by hybridization [39]. Reinforcement learning, another score-based approach, awards or punishes heuristics depending on the improvement or deterioration of the solution. It has been applied for the logistic domain problem [40]. In the combination with tabu search, a tabu list of forbidden heuristics was implemented for nurse rostering and course timetabling [41].

The solution obtained by applying a low-level heuristic might not always be improved. There are different strategies in accepting the solution in case of deterioration. 'All accept' always accepts the solution. Some other strategies accept a new solution with the probability that decreases with time: simulated annealing or Monte Carlo.

In generating new heuristics, one usually involves genetic programming (GP). GP, similarly to evolutionary algorithms, borrows ideas from the theory of natural evolution to automatically produce programs [42]. It starts from a population of generated computer programs which are evaluated by the fitness function. Next, evolutionary components (selection, mutation, crossover) are applied to the individuals in the population and the strongest, i.e. the fittest ones, survive in the next generation. The difference between standard GP and the hyper-heuristic GP is in the generality of the programs used. In the standard approach, the programs could be standard arithmetic operations, standard mathematical functions or logical functions, while in the hyper-heuristic approach, the programs are rather abstract heuristics, independent of the problem domain. As the output from the GP, one gets new programs, which in standard approach could be direct solutions (i.e. modified mathematical formulas or arithmetic operations), but in the hyper-heuristic approach, they need to be translated into solutions. Automatically generated heuristics can be *disposable*, used only once, or *reusable*–can be applied for different instances or problems. In the latter case, generating heuristics are usually trained offline on some benchmark instances.

GP hyper-heuristics were utilized to modify dispatching rules for the job shop scheduling problem ([43], as an example). Grammar-based GP with graph coloring and slot allocation heuristics were applied to exam timetabling [24]. Many applications used GP to evolve heuristics also for the bin packing problem [27, 44] and traveling salesmen problem [25, 45].

There is an interesting connection between the idea of reusable heuristics and transfer learning [46]. In both cases, one may observe the transfer of knowledge between different but related problem domains. However, in the first case, hyper-heuristic is used to tune heuristics from one instance to another or to generate new heuristics, and in transfer learning training data of one problem may be used to potentially improve the results of a target learner on a data set from a different domain. Transfer learning is used mostly in cases when there is a lack of training data or they are too expensive to collect.

4. Hyper-heuristics for bio-inspired combinatorial problems

In this section, a few examples are described in more details for which hyper-heuristic methods were used to solve bio-inspired problems. This field has not been explored deeply, and only a few attempts have been made so far. The difficulty here lies in the quality of the solution. For bio-experts, it is often important that the solution is the optimum or very close to the optimum, not just 'good enough'. Moreover, sometimes mathematical models cannot express biological problems well. The optimization function in the model may lead to a few solutions which are mathematically optimal but only one is biologically correct. Three main contributions from the literature are summarized below, one dealing with the longest common subsequence problem and two with the sequencing by hybridization problem.

4.1. Longest common subsequence

The longest Common Subsequence (LCS) problem amounts to find the longest string that is a subsequence of every string in a given set of strings. The subsequence here is not composed of consecutive letters in the string, but it can be achieved by deleting some of the characters from the string. The problem can be solved polynomially in the case of two strings, but in general, the problem is NP-hard. The example of what the LCS problem is explained in **Figure 3**.

The LCS problem is used in bioinformatics to compare sequences of molecules: DNA, RNA or proteins, in order to find homologies between sequences of organisms of different species. The homology helps to predict the function of unknown genes if their sequences are similar to those of known genes one may expect that the function is similar. The other applications of LCS can be found in text editing [47] or data compression [48], among others.

Tabataba and Mousavi [49] proposed a hyper-heuristic for the LCS problem. A beam search algorithm in its standard form is a tree-based procedure. It starts from the initially empty solution and extends it by one letter in every iteration. All possible characters Σ are evaluated by a function *f(.)* as possible extensions, but only β best ones are further explored in the

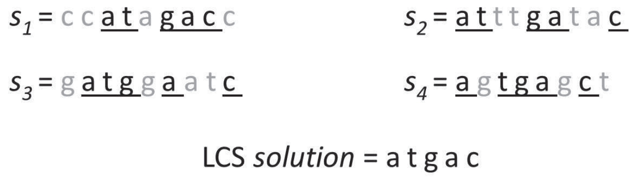
s_1 = c c a t a g a c c
s_2 = a t t t g a t a c
s_3 = g a t g g a a t c
s_4 = a g t g a g c t
LCS *solution* = a t g a c

Figure 3. The example of the LCS problem. For a given set of strings S = {ccatagacc, atttgatac, gatggaatc, agtgagct}, the longest common subsequence of each string is 'atgac'. The LCS is underlined in every string.

next iteration, β being the size of the beam. A hyper-heuristic approach is introduced here to choose the best function *f(.)* among the two available: *power* and *prob*. *Power* takes into account the length of possible suffixes, with the high impact of the minimum suffix, after deciding on the possible extension character. *Prob*, on the other hand, calculates the probability of a random string being a subsequence of the suffix. The hyper-heuristic runs the beam search algorithm twice in every iteration with *power* and *prob* as *f(.)* function and chooses the one which gives the possibility of a longer subsequence extension.

The method was tested on random biologically inspired and real sequences of DNA and proteins of rats and viruses (Σ equal to 4 in the case of DNA sequences and 20 in the case of proteins). Hyper-heuristic appeared to superior to the beam search method used with just one heuristic, either *power* or *prob*. The proposed hyper-heuristic in comparison with the state-of-art algorithms MLCS-APP and DEA, depending on the tested dataset, provides 1–2% and 19–25% improvements in the solution quality, respectively.

4.2. DNA sequencing by hybridization

Sequencing by hybridization (SBH) is a method used for reading DNA sequences, nowadays not used any more due to high costs, but its concepts can be of value for other real-world applications. It is composed of two phases, biological and computational experiments. The former utilizes a microarray chip to determine all the subsequences of an unknown DNA sequence, i.e. subsequences of a given length *k* (*k*-mers). The set of *k*-mers contained in the DNA sequence is called *spectrum*. The latter, combines the elements from the spectrum, by checking their overlapping, into a longer sequence, that do not exceed the original DNA sequence length, *n*. The example of an SBH experiment is shown in **Figure 4**.

In the ideal case, the problem is easy, while in the real experiments, two types of errors may occur which make the real problem NP hard. A negative error occurs when a *k*-mer, being a subsequence of the examined sequence, is missing in the spectrum, while a positive error is an extra element in the spectrum not being a subsequence of the DNA sequence. Note that repeated subsequences in the DNA sequence cause negative errors because they appear only once in the spectrum.

A hyper-heuristic approach was first used to solve the SBH problem in Ref. [39]. The solution is represented as an ordered *list* of elements from the spectrum that contributes to the DNA sequence, and *trash,* an unordered set of 'leftovers' from the spectrum. The sequence is reconstructed with a greedy algorithm that traverses the list and tries to append every *k*-mer with the smallest shift to the preceding one.

The low-level heuristics operate on the list and trash by moving elements from one set to another. In the basic approach, we can distinguish operations on single elements: insertion from trash to list, deletion from list to trash or shift–moves of the elements within the list; or operations on a cluster–a group of closely connected elements. A cluster can be shifted or deleted. An extended approach changed the encoding of the solution, by allowing elements from the spectrum to appear on the list twice, thus solving the problem of repetitions. Also, several new heuristics were proposed, with *swap* as an example.

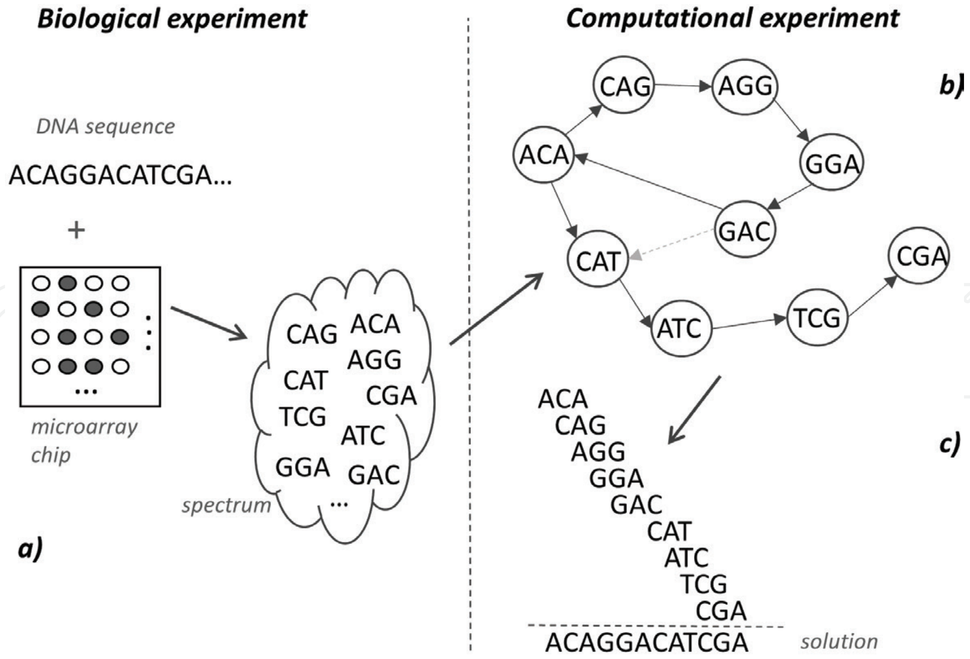

Figure 4. SBH is composed of biological and computational experiments. In the first one (a), a microarray, containing all *k*-mers (*k* = 3), is used to obtain a spectrum. In the latter, elements from the spectrum are modelled as the nodes in the graph in which a Hamiltonian path is looked for (b). Solid arrows represent the overlap of the two nodes equal to 2, and the dashed arrows overlap equal to 1 (most of them are omitted to simplify the picture) meaning that there is a negative error between the two *k*-mers. A path starting from ACA results in obtaining the examined DNA sequence, see the layout in (c). However, notice that starting from node CAG one may obtain a different, shorter solution composed of the same number of elements from the spectrum.

A few hyper-heuristics were proposed, namely a tabu search algorithm, choice function approaches (roulette, ranked, best and decomp), and a simulated annealing algorithm. In the first method, all moves were accepted, while the last method used the Monte Carlo approach which could reject a deteriorated solution with the probability that increased with the passed time. The results of the computational experiment showed that designing a good set of low-level heuristics is very important, a good set could give good results for any tested hyper-heuristics, even for a random roulette choice function, while an incorrectly composed set of primitive heuristics did not allow almost any algorithm to learn which heuristic to choose. The experiments on real DNA sequence instances pointed out two algorithms to be better than others: simulated annealing and the roulette choice function. In the comparison with other algorithms designed for that problem, the usage of elements from the spectrum in the solution was comparable with those obtained by hyper-heuristics, while the similarity of the solution and the examined DNA sequence was superior for tailored-to-measure algorithms.

4.3. DNA sequencing by hybridization, second approach

In Ref. [50], again the DNA sequencing by hybridization problem was considered, but this time accompanied by other combinatorial problems: the knapsack problem, traveling salesman problem (TSP) and its two variants, namely, bottleneck TSP and prize collecting TSP. For these problems, a few hyper-heuristics were implemented, similar to those presented in

Section 4.2 and [39]. It is not new that the same hyper-heuristic is used to solve problems in different domains. The novelty of the proposed approach was in modeling unified encoding of all the above problems, and implementing just one set of low-level heuristics. The heuristics were independent from the problems; thus, a domain barrier was moved down toward problems (compare **Figures 1** and **5**).

The solution for all the problems is represented as sequence *S* of integers from range 1 to n. For SBH, *S* denotes the elements of the spectrum, for TSPs, these are the cities to visit, and for the knapsack problem, *S* denotes the elements to be put to the bin of a given size. A few other data structures or variables were also used: distance matrix, prizes, and penalties, some of them redundant for a given problem, thus not used. The problems could be solved with the hyper-heuristic methods only due to using these specific representations and defining the evaluation function.

The proposed low-level heuristics were simple moves, however, taking into account the specificity of the tested problems. Besides the low-level heuristics previously proposed, insert, delete, swap, and shift, and four more were implemented: (i) replace an element from *S* with an element not being in *S*, (ii) move a few subsequent elements in the solution, (iii) revert the subsequence of elements in S, and (iv) remove the element from *S* which gives the highest cost to the preceding element in the solution, and replace it with the best one.

Some hyper-heuristics could distinguish, useless low-level heuristics or ones leading to unfeasible solutions, and discard them from further computations. The overall results were ‘good enough’, but with the increase of algorithm’s generality, the quality of the obtained solution decreased a little in comparison to hyper-heuristics from Ref. [39] and other tailored metaheuristics.

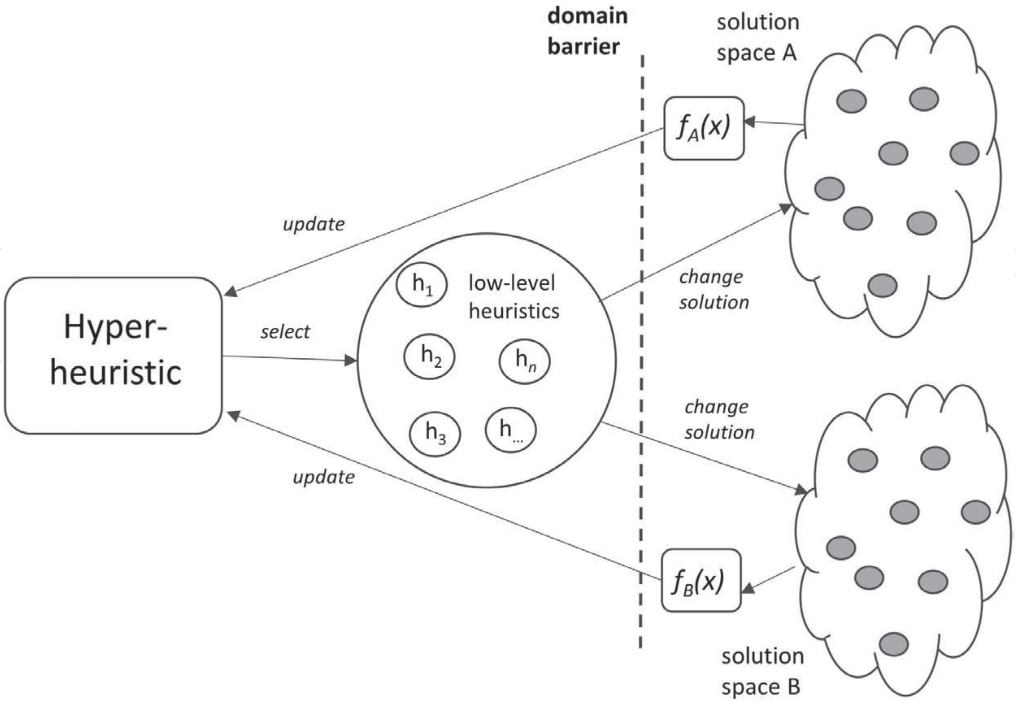

Figure 5. In the unified encoding of the hyper-heuristic scheme there is just one set of low-level heuristics for several problems, while in the standard scheme (**Figure 1**), for each problem, one must implement a separate set of heuristics.

5. Assembling a genome as a continuation of SBH ideas

The previous section presented a few bioinformatics problems solved with hyper-heuristics. Not many attempts have been made so far, due to the specificity of biological problems.

Sequencing by hybridization is a method that did not stand the test of time, but ideas developed in this approach can be translated into the next level of reading DNA sequences, namely assembling. The explosion of new technology allows to directly read short DNA fragments, up to a few hundreds of molecules, and thus making the old-fashioned and costly approaches, Sanger sequencing [51] and SBH, not useful anymore. The inputs to the assembly problem are these longer sequences merged in order to get a longer one of the size of a bacterial genome. In the 'toy problem' -SBH, the fragments are of a length of a dozen or so, and the solution sequence is a few hundred long, while for the assembly problem, DNA fragments are in the range 100–1000 molecules and are assembled into a million-molecule sequence. Both problems can be modeled as searching for a path in the graph, where nodes represent short DNA fragments, and arcs connect overlapping nodes with the cost equal to the shift between the two neighboring fragments. However, in DNA assembly, one must allow mismatches in fragment overlapping, differences in fragment lengths, and the fact that fragments may come from two strands of DNA helix; thus, they are reverse and complementary (the example of the reverse complementary sequence is presented in **Figure 6**). Also, the number of input fragments differs, a few hundred for SBH and millions for assembly. Moreover, during the process of reading DNA fragments, some parts of the genome could be poorly covered by the fragments or even in some cases not covered at all. There might be a few reasons for that, one of them, for example, depends on the content of G and C letters in the fragment of the genome. Thus, the assembly problem becomes much more difficult than SBH, and we cannot say any more about the 'ideal case' and 'easily solvable' problem.

The assembly problem is usually divided into three steps:

Step 1. Finding overlaps of input sequences; constructing a graph.

Step 2. Searching for a path in the graph.

Step 3. Building a consensus sequence.

In the first step, it is usually impossible to calculate the overlaps between each pair of fragments, due to a huge number of fragments and time limitations. In Step 2, instead of one path, the methods output few or more paths, because of sequencing errors, the lack of coverage, and the repetitions in the genome. The last step is the multiple sequence alignment problem, which again is not easily solvable. There are two main approaches to solve the DNA assembly problem. The first one, Overlap-Layout-Consensus (OLC), represents DNA fragments as nodes in the *overlap graph* (Step 1) and calculates in a smart way the overlaps of the fragments. The latter builds a *decomposition-based graph*, not quite precisely called de Bruijn graph, by putting *k*-mers on the nodes and decomposing each DNA fragment into a series of *k*-mers shifted by one. Hence, a DNA fragment, in this case, is represented as a path in the graph connecting a series of respective nodes. In the second case, there is no need to

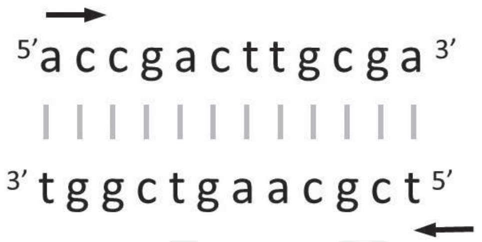

Figure 6. An example of the reverse and complementary sequence. A DNA sequence is always read from 5′ end to 3′ end. Due to the complementarity rule, A on one strand is connected with T on the other, and G is connected with C. The following fragment of the DNA helix can be read as 'accgacttgcga' or 'tcgcaagtcggt'.

calculate overlaps between fragments, because they simply span the same nodes. On the other hand, a lot of information may be lost after decomposing a fragment into *k*-mers. The other two steps are similar for both approaches but take into account the specificity of each approach. There are many methods developed for both of the approaches: OLC [52, 53] and decomposition-based graphs [54, 55] as examples. None of them can be seen as a general process of a local search, where one could use a meta or hyper-heuristic method. Each step of the method is processed independently by a heuristic and/or a greedy approach. However, there is one place where an 'intelligent method', likely a hyper-heuristic, could be of value while searching for a path in the graph. The most difficult in the search process are junctions in the graph, which occur in the case of sequencing errors or repetitions. An example of a junction is presented in **Figure 7**. A hyper-heuristic which could distinguish in the search process that the path is coming to a junction and cut the current process would highly improve the solution. The method shall take into account the increase or decrease of the coverage and basing on this change and decide whether to stop or continue the search process. This must be preceded by offline training of many benchmark data sets, where a method could learn the specific cases in the graph and make predictions in the future search. The two basic steps that

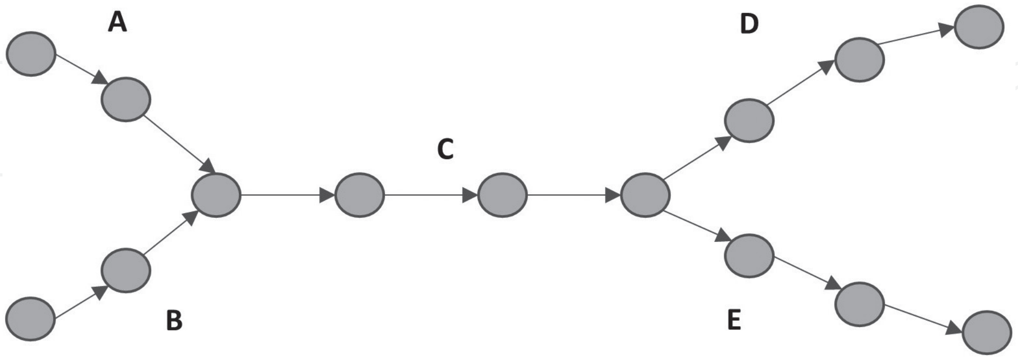

Figure 7. Junctions in the graph complicate the assembly process. In order to simplify the figure, only the arcs with the smallest shift between the two nodes are given. Without additional information, it is difficult to state if the correct sequence should be composed of fragments ACD, ACE, BCD, or BCE. Thus, the methods usually cut the current path and output shorter fragments like AC or CE.

a hyper-heuristic would choose could be a simple ‘stop searching’ and ‘continue extending the path’, but of course, many other heuristics like extending search path in both directions could also be added.

One may notice that the field of bioinformatics combinatorial optimization problems is not well explored in terms of hyper-heuristic search. The assembly problem is just one example of many problems, where a hyper-heuristic could be introduced. The question is if we are ready to allow a possible deterioration of the final solution by generalizing the search process. The other question is what is the most time-consuming activity: understanding a bio problem, designing a model, or implementing a method? In the case of some problems, it is crucial to clearly understand the problem because omitting some of the details makes the solution unacceptable for bio-experts. Hyper-heuristics and especially low-level heuristics should be developed to comply with all the restrictions and produce a feasible solution, which, unfortunately, in many cases may take as much time as implementing a ‘tailored-to-measure’ algorithm.

6. Summary and future

The flow of ideas between biology, operational research, and computer science works in both directions. The ideas from the biological evolution serve as the basis of genetic algorithms and genetic programming. The operational research models and methods help to solve many problems arising in computational biology. Recently, an interesting cooperation between these two scientific areas has been observed in the behavior of the slime mold *Physarum*, for which a mathematical model for the dynamics always converges in finding the shortest path for any input graph [56, 57]. There were also some attempts to involve DNA and use it as the computing power [58].

We can observe a constant synergy between specialists from different areas. This synergy can also be found in the context of hyper-heuristic methodology. It has been shown that hyper-heuristics have been successfully involved in solving several combinatorial optimization problems. Also, a few attempts have been made to solve bio-inspired problems: the longest common subsequence problem and sequencing by hybridization problem.

In Section 5, a proposition how to employ hyper-heuristic search also in solving the DNA assembly problem has been made. By allowing offline learning on some benchmark dataset, the method could distinguish the junctions of the path in the graph and react faster whether to extend or cut the current path.

A good learning mechanism incorporated into hyper-heuristics is a key to increase the usage of hyper-heuristics and to solve the problems in a competitive way to tailored-to-measure (meta)heuristics. In this context, the development of the tools for assessing hyper-heuristics such as HyFlex [59] or Hyperion [60] may increase the usage of these types of methods.

Acknowledgements

The author would like to thank J. Blazewicz, G. Felici and the anonymous referee for valuable remarks and the discussions which significantly improved the presentation of the paper.

The research was partially supported by a Poznan University of Technology grant [09/91/DSPB/0600].

Author details

Aleksandra Swiercz

Address all correspondence to: aswiercz@cs.put.poznan.pl

1 Institute of Computing Science, Poznan University of Technology, Poland

2 Institute of Bioorganic Chemistry, Polish Academy of Sciences, Poznań, Poland

References

[1] Cowling P, Kendall G, Soubeiga E. A hyperheuristic approach for scheduling a sales summit. In: Selected Papers of the 3rd International Conference on the Practice and Theory of Automated Timetabling, PATAT 2000. Berlin: Springer; 2001. pp. 176-190

[2] Fisher H, Thompson GL. Probabilistic learning combinations of local job-shop scheduling rules. In: Muth JF and Thompson GL, editors. Industrial Scheduling. NY: Prentice-Hall: Englewood Cliffs; 1963. pp. 225-251

[3] Storer RH, Wu SD, Vaccari R. New search spaces for sequencing problems with application to job shop scheduling. Management Science. 1992;**38**(10):1495-1509

[4] Fang H, Ross P, Corne D. A promising hybrid ga/heuristic approach for openshop scheduling problems. In: Cohn AG, editor. European Conference on Artificial Intelligence. New York: John Wiley & Sons; 1994

[5] Drechsler R, Becker B. Learning heuristics by genetic algorithms. In: Shirakawa I, editor. ASP Design Automation Conference. ACM: Makuhari, Massa, Chiba, Japan; 1995. pp. 349-352

[6] Grefenstette J. Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics SMC. 1986;**16**(1):122-128

[7] Bäck T. An overview of parameter control methods by self-adaption in evolutionary algorithms. Fundamental Journals. 1998;**35**(1-4):51-66

[8] Rice JR. The algorithm selection problem. Advances in Computers. 1976;**15**:65-118

[9] Sleeman D, Rissakis M, Craw S, Graner N, Sharma S, Consultant-2: Pre–and post-processing of machine learning applications. International Journal of Human Computer Studies. 1995;**43**(1):43-63

[10] Wah BW, Ieumwananonthachai A. Teacher: A genetics-based system for learning and for generalizing heuristics. In: Yao X, editor. Evolutionary Computation. Singapore: World Scientific Publishing Co. Pte. Ltd.; 1999. pp. 124-170

[11] Burke EK, Gendreau M, Hyde M, Kendall G, Ochoa G, Ozcan E, Qu R. Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society. 2013;**64**:1695-1724

[12] Pappa GL, Ochoa G, Hyde MR, Freitas AA, Woodward J, Swan J. Genetic Programming and Evolvable Machines. Contrasting meta-learning and hyper-heuristic research: The role of evolutionary algorithms. 2014;**15**(1):3-35. DOI: 10.1007/s10710-013-9186-9

[13] Lysov Y, Florent'ev V, Khorlin A, Khrapko K, Shik V, Mirzabekov A, Determination of the nucleotide sequence of DNA using hybridization with oligonucleotides. A new method. Doklady Akademii Nauk SSSR. 1988;**303:**1508-1511

[14] Pevzner P, l-tuple DNA sequencing: Computer analysis. Journal of Biomolecular Structure and Dynamics. 1989;**7:**63-73

[15] Blazewicz J, Hertz A, Kobler D, de Werra D. On some properties of DNA graphs. Discrete Applied Mathematics. 1999;**98:**1-19

[16] Vrugt J, Robinson B. Improved evolutionary optimization from genetically adaptive multimethod search. Proceedings of the National Academy of Sciences. 2007;**104**(3):708-711

[17] Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E, Woodward J. Handbook of metaheuristics, international series in operations research & management science. Vol. 146. Chap. A Classification of Hyper-heuristic Approaches, New York: Springer; 2010. pp. 449-468. Chapter 15

[18] Ahmadi S, Barrone P, Cheng P, Burke EK, Cowling P, McCollum B. Perturbation based variable neighbourhood search in heuristic space for examination timetabling problem. In: Kendall G, Burke EK, Petrovic S, Gendreau M, editors. Multidisciplinary International Scheduling: Theory and Applications. New York: MISTA, Springer; 2003. pp. 155-171

[19] Burke EK, McCollum B, Meisels A, Petrovic S, Qu R. A graph-based hyperheuristic for educational timetabling problems. European Journal of Operational Research. 2007; **176**:177-192

[20] Sabar NR, Ayob M, Qu R, Kendall G. A graph coloring constructive hyper-heuristic for examination timetabling problems. Applied Intelligence. 2012;**37:**1-11

[21] Cano-Belmán J, Ríos-Mercado R, Bautista J. A scatter search based hyperheuristic for sequencing a mixed-model assembly line. Journal of Heuristics. 2010;**16**:749-770

[22] Dowsland KA, Soubeiga E, Burke EK. A simulated annealing hyperheuristic for determining shipper sizes for storage and transportation. European Journal of Operational Research. 2007;**179**(3):759-774

[23] Pisinger D, Ropke S. A general heuristic for vehicle routing problems. Computers and Operations Research. 2007;**34**(8):2403-2435

[24] Pillay N. Evolving hyper-heuristics for the uncapacitated examination timetabling problem. In: Blazewicz J, Drozdowski M, Kendall G, McCollum B (eds). Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA'09). Dublin, Ireland; 2009. pp. 447-457

[25] Keller RE, Poli R. Cost-benefit investigation of a genetic-programming hyperheuristic. In: Monmarche' N, Talbi E-G, Collet P, Schoenauer M, Lutton E, editors. International Conference on Artificial Evolution. Berlin, Heidelberg: Springer-Verlag; 2007. pp. 13-24

[26] Runka A. Evolving an edge selection formula for ant colony optimization. In: Genetic and evolutionary computation conference (GECCO'09). New York: ACM; 2009. pp. 1075-1081

[27] Burke EK, Hyde MR, Kendall G. Evolving bin packing heuristics with genetic programming. In: Parallel Problem Solving from Nature PPSN IX, Lecture Notes in Computer Science. Runarsson T.P., Beyer HG, Burke E, Merelo-Guervós JJ, Whitley LD, Yao X (eds). Springer, Berlin, Heidelberg. Vol. 4193. 2006. pp. 860-869

[28] Burke EK, Hyde MR, Kendall G. Grammatical evolution of local search heuristics. IEEE Transactions on Evolutionary Computation. 2012;**16**:406-417

[29] Sim K, Hart E, Paechter B. A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by attribute evolution. In: Parallel Problem Solving from Nature: PPSN XII, Vol. 7492. Coello CAC, Cutello V, Deb K, Forrest S, Nicosia G, Pavone M. (eds). Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. 2012. pp. 348-357

[30] Ross P, Marín-Blázquez JG. Constructive hyper-heuristics in class timetabling. In: IEEE Congress on Evolutionary Computation (CEC'05), Edinburgh, UK. 2005. pp. 1493-1500

[31] Burke EK, Petrovic S, Qu R. Case based heuristic selection for timetabling problems. Journal of Scheduling. 2006;**9**(2):115-132

[32] Fukunaga AS. Automated discovery of local search heuristics for satisfiability testing. Evolutionary Computation. 2008;**16**(1):31-61

[33] Burke EK, Hyde MR, Kendall G, Woodward J. Automating the packing heuristic design process with genetic programming. Evolutionary Computation. 2012;**20**(1):63-89

[34] Krasnogor N, Gustafson S. A study on the use of "self-generation" in memetic algorithms. Natural Computing. 2004;**3**(1):53-76

[35] Garrido P, Riff M. Dvrp: A hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. Journal of Heuristics 2010;**16**:795-834

[36] Remde S, Cowling P, Dahal K, Colledge N. Binary exponential back-off for tabu tenure in hyperheuristics. In: Cotta C, Cowling P, editors. Evolutionary Computation in Combinatorial Optimization. Vol. 5482. Berlin: Springer; 2009. pp. 109-112

[37] Terashima-Marín H, Ross P, Farías-Zárate CJ, López-Camacho E, Valenzuela-Rendón M. Generalized hyper-heuristics for solving 2D regular and irregular packing problems. Annals of Operations Research 2010;**179**(1):369-392

[38] Rattadilok P, Gaw A, Kwan RSK. Distributed choice function hyper-heuristics for timetabling and scheduling. In: Burke EK, Trick M, editors. The Practice and Theory of Automated Timetabling V: Selected Papers from the 5th International Conference on the Practice and Theory of Automated Timetabling. Vol. 3616. Lecture Notes in Computer Science Series. Berlin: Springer; 2005. pp. 51-70

[39] Blazewicz J, Burke EK, Kendall G, Mruczkiewicz W, Oguz C, Swiercz A. A hyper-heuristic approach to sequencing by hybridization of DNA sequences. Annals of Operations Research. 2013;**207**(1):27-41. DOI: 10.1007/s10479-011-0927-y

[40] Nareyek A. An empirical analysis of weight-adaptation strategies for neighborhoods of heuristics. In: Sousa J, editor. Metaheuristic International Conference MIC'2001. Porto, Portugal. 2001. pp. 211-215

[41] Burke EK, Kendall G, Soubeiga E. A tabu-search hyperheuristic for timetabling and rostering. Journal of Heuristics. 2003;**9**(6):451-470

[42] Koza JR. Genetic Programming: On the Programming of Computers by Means of Natural Selection. Boston, MA: The MIT Press; 1992

[43] Ho NB and Tay JC. Evolving dispatching rules for solving the flexible job-shop problem. In: IEEE Congress on Evolutionary Computation (CEC'05). Edinburgh, UK: IEEE; 2005. pp. 2848-2855

[44] Poli R, Woodward JR and Burke EK. A histogram matching approach to the evolution of bin-packing strategies. In: IEEE Congress on Evolutionary Computation (CEC'07). Singapore: IEEE; 2007. pp. 3500-3507

[45] Oltean M, Dumitrescu D. Evolving TSP heuristics using multi expression programming. In: International Conference on Computational Science (ICCS'04). Vol. 3037. Lecture Notes in Computer Science. Berlin: Springer; 2004. pp. 670-673

[46] Weiss K, Khoshgoftaar TM, Wang DD. A survey of transfer learning. Journal of Big Data. 2016;**3**:9

[47] Sankoff D, Kruskal J. Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparisons. Addison-Wesley. 1983

[48] Storer JA. Data Compression: Methods and theory. Computer Science Press Inc.; New York, NY, USA. 1988

[49] Tabataba FS, Mousavi SR. A hyper-heuristic for the longest common subsequence problem. Computational Biology and Chemistry. 2012;**36**:42-54

[50] Swiercz A, Burke EK, Cicheński M, Pawlak G, Petrovic S, Zurkowski T, Blazewicz J. Unified encoding for hyper-heuristics with application to bioinformatics. Central European Journal of Operations Research. 2014;**22:**567-589

[51] Sanger F, Nicklen S, Coulson A. DNA sequencing with chain-terminating inhibitors, Proceedings of the National Academy of Sciences, USA. 1977;**74:**5463-5467

[52] Myers E, Sutton G, Delcher A. A whole-genome assembly of Drosophila, Science. 2000; **287**(5461):2196-2204

[53] Simpson J, Durbin R. Efficient de novo assembly of large genomes using compressed data structures, Genome Research. 2012;**22**:549-556

[54] Zerbino D, Birney E. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs, Genome Research. 2008;**18**:821-829

[55] Kajitani R, Toshimoto K, Noguchi H, Toyoda A, Ogura Y, Okuno M, Yabana M, Harada M, Nagayasu E, Maruyama H, Kohara Y, Fujiyama A, Hayashi T, Itoh T. Efficient de novo assembly of highly heterozygous genomes from whole-genome shotgun short reads, Genome Research. 2014;**24**:1384-1395

[56] Bonifaci V, Mehlhorn K, Varma G. Physarum can compute shortest paths. Journal of Theoretical Biology. 2012;**309**:121-133

[57] Bonifaci V. Physarum can compute shortest paths: A short proof. Information Processing Letters. 2013;**113**(1-2):4-7

[58] Adleman LM. Molecular computation of solutions to combinatorial problems. Science. 1994;**266**:1021-1024

[59] Ochoa G, Walker J, Hyde M, Curtois T. Adaptive evolutionary algorithms and extensions to the HyFlex hyper-heuristic framework. In: Parallel Problem Solving from Nature—PPSN XII. Lecture Notes in Computer Science. Berlin: Springer. 2012;**7492**:418-427

[60] Swan J, Özcan E, Kendall G. Hyperion—A recursive hyper-heuristic framework. In: Coello CAC, editor. LION. Lecture Notes in Computer Science. Berlin: Springer. 2011;**6683**:616-630