CEJOR (2014) 22:567–589
DOI 10.1007/s10100-013-0321-8

ORIGINAL PAPER

Unified encoding for hyper-heuristics with application to bioinformatics

Aleksandra Swiercz · Edmund K. Burke · Mateusz Cichenski · Grzegorz Pawlak · Sanja Petrovic · Tomasz Zurkowski · Jacek Blazewicz

Published online: 8 August 2013

Abstract This paper introduces a new approach to applying hyper-heuristic algorithms to solve combinatorial problems with less effort, taking into account the modelling and algorithm construction process. We propose a unified encoding of a solution and a set of low level heuristics which are domain-independent and which change the solution itself. This approach enables us to address NP-hard problems and generate good approximate solutions in a reasonable time without a large amount of additional work required to tailor search methodologies for the problem in hand. In particular, we focused on solving DNA sequencing by hybrydization with errors, which is known to be strongly NP-hard. The approach was extensively tested by solving multiple instances of well-known combinatorial problems and compared with results generated by meta heuristics that have been tailored for specific problem domains.

Keywords Bioinformatics · Hyper-heuristics · Simulated annealing · Choice function · Combinatorial problems · Sequencing by hybrydization

A. Swiercz · M. Cichenski (✉) · G. Pawlak · T. Zurkowski · J. Blazewicz
Institute of Computing Science, Poznan University of Technology, Poznan, Poland
e-mail: mateusz.cichenski@cs.put.poznan.pl

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

E. K. Burke
University of Stirling, Stirling, UK

S. Petrovic
University of Nottingham, Nottingham, UK

1 Introduction

In recent years, more and more biologically inspired problems have been studied by operational researchers. The cooperation between these two disciplines demonstrates that a strong scientific synergy between molecular biology and operational research problems can be established. Many methods have been developed to cope with the large amount of experimental data and new tools have been created to predict the structure of molecules. A prime example of a biological problem is reading deoxyribonucleic acid (DNA) sequences. It cannot be done at once due to the fact that the genome sequence is simply too long. For example: a genome of bacteria *Escherichia coli* is composed of 4.6×10^6 small molecules, called nucleotides, a genome of yeast has 1.4×10^7 and a genome of wheat is twice as large as the one from a human, which is 3.3×10^9. Usually, the process of reading DNA is divided into three phases: sequencing short fragments of DNA, assembling these short fragments into longer ones, and then placing them into the proper places in the chromosome. In the sequencing part, one can read 50–600 nucleotide long sequences at once, depending on the method employed. One such method is called sequencing by hybridization (SBH) (Southern 1988). It is based on a biochemical experiment which basically determines the subfragments of the examined DNA sequence. Naturally, in order to obtain the original sequence, the subfragments which overlap must be merged together. This can be done by combinatorial algorithms. There have been many successful solutions already presented in the literature (Lysov et al. 1988; Dramanac et al. 1989; Pevzner 1989; Bui and Youssef 2004; Blazewicz and Kasprzak 2003; Blazewicz et al. 1997, 2002, 2004, 2006a,b).

This paper presents a unique insight into the DNA sequencing by hybridization problem from the perspective of developing generic search techniques which could be applied to different combinatorial optimization problems. Our aim is to explore the generality of hyper-heuristics, namely whether a hyper-heuristic can solve the DNA sequencing by hybridization (SBH) problem in addition to some selected well-known combinatorial optimization problems. In our research study, we chose these optimization problems: traveling salesman problem, bottleneck traveling salesman problem (TSP), prize collecting TSP, and knapsack problem. One of the main motivating ideas behind hyper-heuristics is to use the same search methodology across different problem domains whilst minimizing the manual adaptation of the search algorithm.

We focus on a hyper-heuristic framework which consists of a set of heuristics, also called low level heuristics and a hyper-heuristic algorithm, also called high level heuristic. The latter one evaluates the performance of low level heuristics and selects one of them, which is then applied to change current solution (Burke et al. 2003a, 2013; Ross et al. 2003; Ross 2005). The performance can be measured as an increase of the objective function value, which is defined by the problem. However, one can also take into account other factors, e.g. the time of computation or the number of successful applications of the given low level heuristics. The process is repeated iteratively, i.e. a low level heuristic is selected from among the available heuristics in the set and applied to change the solution. The behavior depends on the hyper-heuristic algorithm itself. However, it enables jumping out of a local optimum.

Thus, these hyper-heuristic approaches operate on the low level heuristic search space and try to find the best execution order of the available heuristics. It may be necessary to provide a different set of heuristics to solve different problem domains, but the hyper-heuristic is kept untouched. In contrast, most implementations of meta heuristic approaches conduct a search over a representation of the problem and are usually considered as made-to-measure, algorithms "tailored" to the needs of the given problem domain (Dowsland 1998) with respect to the problem specific parameters, constraints etc. However, developing specialized algorithms for each problem domain is usually time consuming and requires a lot of parameter tuning to provide acceptable solutions. Hyper-heuristic approaches can adapt themselves to the problem at hand, operating on the supplied low level heuristics set, yet giving satisfactory results with less "problem tailoring" required (Burke et al. 2003b).

Hyper-heuristic algorithms have been successfully employed in solving many problem domains. We investigate two meta-heuristics to be used as high level heuristics in the hyper-heuristic framework: simulated annealing (Kirkpatrick et al. 1983; Aarts et al. 2005), tabu search (Glover and Laguna 1997; Gendrau and Potvin 2005) and also the choice function (Cowling et al. 2001, 2002a). There are also new approaches that use multiple phases (Cichowicz et al. 2012a) or genetic algorithms (Cichowicz et al. 2012b). Hyper-heuristics inspired by simulated annealing have been considered for problems such as automating the design of supermarket shelf layouts (Bai and Kendall 2005) and determining shipper sizes (Dowsland et al. 2007). Tabu search based hyper-heuristics have been applied to different problems including timetabling and rostering (Burke et al. 2003b, 2007; Kendall and Hussin 2005). Lastly, choice function hyper-heuristics have been used on a sales summit scheduling problem (Cowling et al. 2001), scheduling project presentations (Cowling et al. 2002a) and the nurse rostering problem (Cowling et al. 2002b). However, these approaches were mostly developed and evaluated on a single optimization problem.

In this paper, we propose a unified encoding for the selected optimization problems to be used in the hyper-heuristic framework. The aim of the unified encoding is to simplify the process of constructing an effective search procedure by omitting the need to create complex low level heuristics or customizing meta heuristics. Simple low level heuristics are used on the proposed unified encoding instead. Additionally, the aim of the study is to show that such a unified representation is possible, even for problems coming from different research domains including bioinformatics. We run extensive experiments to evaluate the unified encoding and performance of the hyper-heuristic on the selected optimization problems. The results were compared to meta heuristics and hyper-heuristics which were not as generic as our approach, to verify if the generalization of the methodology can still yield satisfactory results. It is not to be expected that the generalised approach produces better results than the tailored one (although it may happen). However, a good performance of the hyper-heuristic will open the door to the investigation of their applicability to other combinatorial optimization problems including bioinformatics ones, which could be represented by the proposed unified encoding. It would mean that a variety of other bioinformatics problems could be solved with less effort with respect to the time needed for the development and fine tuning of the algorithm.

The paper is organized as follows. The next section provides the definitions of investigated combinatorial problems. Section 3 describes the idea of hyper-heuristics in more detail including an explanation of choice function based hyper-heuristics, and

Sect. 4 contains the description of the new approach proposed here. In Sect. 5 the low level heuristics used in the experiments are described. The computational experiments are presented in Sect. 6. Final remarks are included in Sect. 7.

2 Combinatorial problems formulation

Five combinatorial problems were employed to test the proposed approach. Four of them were investigated during the selection of the representation and one was used as a hidden domain to test the power of the approach. The following subsections provide short descriptions of the problems with special attention paid to the SBH problem.

2.1 Sequencing by hybridization

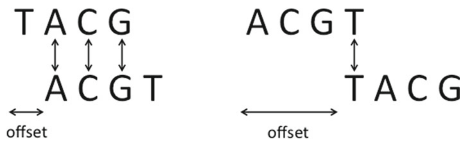
As mentioned above, during biochemical experiments it is not possible to read the whole DNA sequence of a genome at once, because it is too long (about 3.3×10^9 nucleotides for humans). Instead, biochemists are able to determine short fragments of a given length l, which are subsequences of the examined DNA sequence. These fragments are sequences of nucleotides represented by letters A, C, G and T. The letters correspond to the nitrogenous bases contained in the nucleotides: adenine, cytosine, guanine and thymine, respectively. Next, the short fragments have to be merged together into one, longer DNA sequence. The method that determines the order of nucleotides is called SBH.

Usually, it is not possible to determine all the subfragments of the examined DNA sequence. On the other hand, sometimes fragments determined during the experiment are not the subfragments of the DNA sequence. These missing and erroneous fragments make the reconstruction of the DNA sequence difficult. The SBH problem in the case of errors in the set of input fragments belongs to the strongly NP-hard class of problems.

Given a set W of words (l-mers) of length l over the alphabet $\{A, C, G, T\}$ and length m of the original DNA sequence, one has to find the unknown original DNA sequence, not longer than m, which will contain the maximum number of l-mers from W. The words can overlap with an offset, e.g. word $ACGT$ overlaps word $TACG$ with offset 1, but word $TACG$ overlaps word $ACGT$ with offset 3 (because word $TACG$ can start at the third position of word $ACGT$ without any misplaced letter). This example can be analyzed in Fig. 1.

The SBH problem can be represented as a graph problem—l-mers are represented as vertices and the distance between two nodes is the offset between these two l-mers. The task is to find a path that visits the maximum number of nodes and its length is less than or equal to $Q = m - l$.

Fig. 1 An example of l-mers overlapping



2.2 The traveling salesman problem

The TSP is strongly NP-hard. Given a list of cities and distances between them, the task is to find the shortest possible route that visits each city exactly once. Richard Karp has proved that the determination of a Hamiltonian cycle is an NP-complete problem, which implies that TSP is NP-hard (Karp 1972).

Given n cities and $n \times n$ matrix D of distances between each pair of cities the task is to find the shortest path, which will allow a visit to all the cities and which will let the salesman come back to the city from which the path started, i.e. the shortest cycle visiting all cities.

The problem can also be represented by using a directed graph. Each city is represented by a node. There is an arc between two nodes i and j if there is a direct route from city i to city j. The weight of the arc represents the distance between those cities. The task is to find the shortest Hamiltonian cycle.

2.3 Bottleneck traveling salesman problem

This problem is similar in formulation to TSP, but it has a different objective—one has to find a cycle (path) that will go through all the cities and minimize the weight of the most costly connection between the cities (or arc in graph representation) in the cycle (path).

2.4 The prize collecting traveling salesman problem

In addition to the TSP definition, a prize Pr_i for visiting city i and a penalty Pe_i for not visiting a city is introduced in this problem. The minimum amount Q of prizes that has to be collected is also defined. The goal is to find a cycle that minimizes the total route length and the sum of penalties for not visited cities while collecting at least a specified amount of prizes. In contrast to classic TSP, one does not have to visit all the cities.

2.5 The knapsack problem

The knapsack problem is an NP-hard optimization problem where, for given items with associated weights and values, one has to pack them into one knapsack, that can hold items up to some total maximal weight. Hereafter, the 0–1 knapsack problem will be investigated, in which every item can be used at most once. This problem was not considered during the selection of the unified encoding. It was a hidden domain that was used to evaluate the approach on the problem that can be represented by the proposed model, but was not designed for it.

Given n items, with weights w_i and value v_i, $i = 1 \ldots n$, the task is to find a subset of items that will fit to a knapsack with a weight limit Q and will maximize the total value of items inside the knapsack.

3 Hyper-heuristic approaches

Hyper-heuristics can be described as a knowledge-poor approach (Cowling et al. 2001; Özcan et al. 2008) to solve problems in different domains. An overview of hyper-heuristic methods can be found in Burke et al. (2013). We can consider the role of the hyper-heuristic to be to choose one or more low level heuristics from a given set and apply them iteratively to improve the resulting solution, in other words selecting appropriate existing heuristics (Burke et al. 2010). Those low level heuristics are specialized in solving certain problem domains, but the ideal hyper-heuristic does not have knowledge about the problem itself—it uses only the quality measures of the performance of low level heuristics which are independent from the problem domain, e.g. how many times the heuristics improved solution and by what factor. Thus, given a hyper-heuristic framework one has to define only the set of low level heuristics and one or more quality measures to rate the solutions (Cowling et al. 2002b).

In 1975 John Rice proposed a framework for the algorithm selection problem (Rice 1976) which can be used to define hyper-heuristic algorithms. This framework seeks which algorithm from a given set is likely to perform best, based on measures derived from a collection of problem instances. The framework consists of four components:

– Problem space $\mathscr{P}$ which is represented by a set of instances of a problem
– Feature space $\mathscr{F}$ which contains measurable characteristics of the instances from $\mathscr{P}$
– Algorithm space $\mathscr{A}$ which is a set of considered algorithms that can be used to solve the problem
– Performance space $\mathscr{Y}$ which contains performance metrics of each algorithm for a given problem instance

The algorithm selection problem is stated in Smith-Miles and Lopes (2012) as: For a given problem instance $x \in \mathscr{P}$, with feature vector $f(x) \in \mathscr{F}$, find the selection mapping $S(f(x))$ into algorithm space $\mathscr{A}$ such that the selected algorithm $\alpha \in \mathscr{A}$ maximizes the performance metric $\|y\|$ for $y(\alpha, x) \in \mathscr{Y}$. The collection of data describing $\{\mathscr{P}, \mathscr{A}, \mathscr{Y}, \mathscr{F}\}$ is known as meta-data.

Thanks to Rice's framework one can define the hyper-heuristic algorithm as follows: it is possible to map the performance of low level heuristics as feature space $\mathscr{F}$ as opposed to calculating the characteristics of the problem instances. In other words, the performance space $\mathscr{Y}$ is the same as feature space $\mathscr{F}$ when considering a hyper-heuristic approach (Burke et al. 2003a). Unfortunately, Rice did not clarify how the feature space should be calculated and this is widely explored in Smith-Miles and Lopes (2012). Even using performance space $\mathscr{Y}$ as features does not provide knowledge about how the mapping into algorithm space $\mathscr{A}$ should be undertaken. This is the task of designing hyper-heuristic algorithms.

3.1 How hyper-heuristics work

In general, the input data for our hyper-heuristic framework contains a set of low level (problem specific) heuristics and one or more quality measures (domain barrier) to rate the produced solutions. Thus, the single hyper-heuristic algorithm can be employed

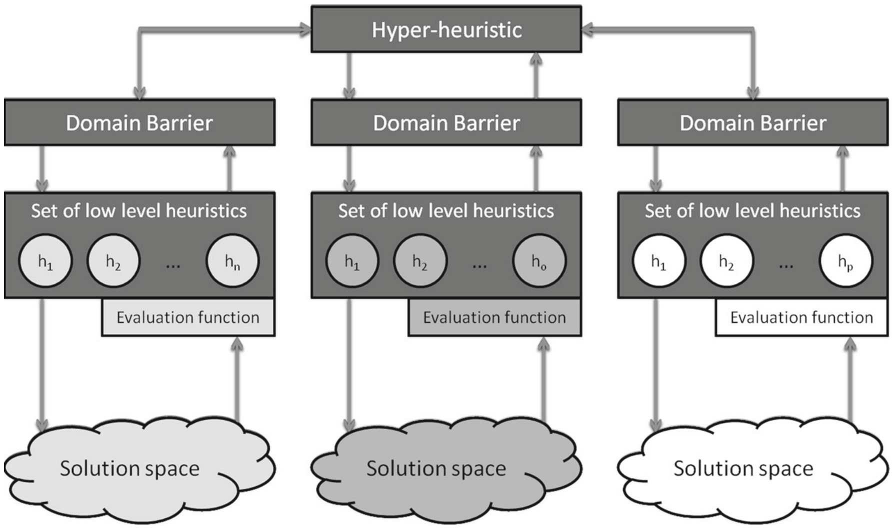


Fig. 2 A standard hyper-heuristic approach [similar to that presented in Cowling et al. (2001)]

to solve different problem domains, which is illustrated in Fig. 2. The process is iteratively going through the following steps:

1. Find the starting solution and set it as current
2. Apply one or more low level heuristics to the solution and measure the quality of the new solution
3. Select one of the solutions from 2 and set it as current
4. If stop conditions are not satisfied go to step 2, otherwise STOP

In step 1, any initial solution can be used. However, it is important to notice that different starting solutions may provide different final results (if they are not optimal) due to the fact that a different part of solution space could be explored. In case of TSP, the initial solution can be set to a solution generated by a random number permutation or to a more complex greedy heuristic.

Step 2 is the most important, as it defines the behavior of the hyper-heuristic. There are many possible approaches inspired by meta heuristic.

Most commonly hyper-heuristics have, as a termination criterion, a time limit, a number of iterations without significant improvement in the solution quality or a number of iterations the algorithm can perform. In practice, the termination condition used depends on the constraints given by the user—if the problem should be solved within a given time period a time limit should be used, if time is not so important and the goal is to obtain the best possible result, the number of iterations without significant improvement can be used.

3.2 Choice function

One of the possibilities to create a hyper-heuristic algorithm is to base it on mathematical functions, e.g. the choice function (Cowling et al. 2001). This method uses three components—the effectiveness of the low level heuristic, the effectiveness of a pair of low level heuristics (how well two heuristics work together) and the part responsible for the diversification of the solution (when the heuristic was last used).

Let us assume that I_i is the improvement in objective function reached by heuristic h in iteration i (if it was not used in iteration i then $I_i = 0$), T_i is the time taken by the low level heuristics used in iteration i and t is the current iteration. The first component takes into account only the performance of a single heuristic, and can be calculated as:

$$c_1(h) = \sum_{i=1}^{t} \alpha^{t-i} \frac{I_i}{T_i} \tag{1}$$

where α, $0 < \alpha < 1$, is the parameter used to weight the past results. This function uses the performance measure. It also employs the time required to execute the low level heuristic.

The second component is used to evaluate the performance of one solution occurring after another. It is possible that a pair of low level heuristics will work well together and it is worth applying first heuristic and then immediately using the other one after it. To calculate this value the following formula is used:

$$c_2(h) = \sum_{i=1}^{t} \beta^{t-i} \frac{Ip_i}{T_i} \tag{2}$$

where g is a heuristic that was used in the previous iteration, and Ip_i is the improvement in the objective function made by heuristic h in iteration i, but only if it was used after g. For each iteration i when h was not used right after g let $Ip_i = 0$. β is a weight that can be tuned for every problem separately, $0 < \beta < 1$. The difference between c_1 and c_2 is in iterations that are considered—in c_1 all iterations when heuristic h was used are taken into consideration and in c_2 only those iterations when h was preceded by g are used.

The last component is equal to the number of seconds elapsed from the last usage of a given low level heuristic $t(h)$ scaled by factor γ. This value helps to diversify the search—if the heuristic was not used for a long time, the value will be higher.

$$c_3(h) = \gamma t(h) \tag{3}$$

The choice function for a given heuristic h is equal to:

$$c(h) = c_1(h) + c_2(h) + c_3(h) \tag{4}$$

The positive choice function for a given heuristic is defined as:

$$c^+(h) = \exp\left[\eta c(h)\right] \tag{5}$$

where η is a scaling factor.

The final choice function and its components can be used to create different hyper-heuristic algorithms.

Four hyper-heuristics based on the choice function are defined below, including straight choice, ranked choice, decomp choice and roulette choice (Cowling et al. 2001; Mruczkiewicz 2009). They differ from one another in the way they choose the heuristic that should be used in a given iteration.

3.2.1 Straight choice

The first algorithm, straight choice, always chooses the heuristics with the highest value of the choice function $c(h)$. Only the value of choice function is used to determine the heuristics chosen in a given iteration. This means that in this method only one heuristic is executed in every iteration.

3.2.2 Ranked choice

The ranked choice algorithm selects k heuristics with the highest choice function (k is defined for every problem separately). Then it applies these heuristics to a current solution and compares the objective function of a new obtained solution. The heuristic that achieved the best objective function is chosen and used in the next iteration.

This algorithm requires k runs of the low level heuristics to choose one that will be used in a given iteration.

3.2.3 Decomp choice

The decomp choice algorithm selects four heuristics—with the best value of $c_1(h), c_2(h), c_3(h), c(h)$. Next, each of those four heuristics is applied to a current solution, and the one that achieved the best objective function is chosen in a given iteration.

This algorithm requires exactly four runs of low level heuristics.

3.2.4 Roulette choice

The roulette choice based hyper-heuristic chooses the heuristics randomly with the probability of choosing a given heuristics h equal to:

$$P(h) = \frac{c^+(h)}{\sum_{g \in H} c^+(g)} \tag{6}$$

4 Unified encoding approach

Solving a problem domain using a standard hyper-heuristic approach requires defining the set of low level heuristics, the objective function and the solution representation. The last two parts are described by problem formulation. However, the low level heuristics are the hardest part, because one has to know the characteristics of the

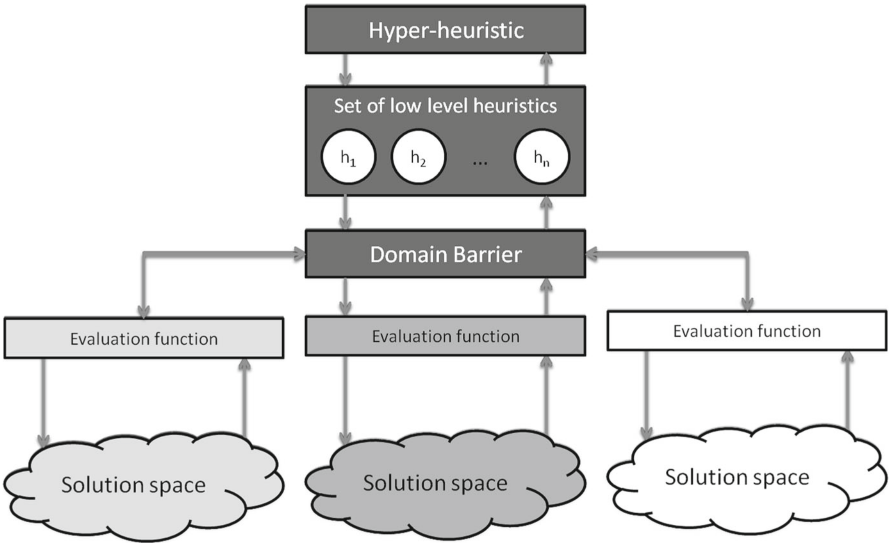


Fig. 3 Unified encdoding hyper-heuristic approach

problem. For example, a good low level heuristic for the TSP would be the well known 2-OPT neighborhood (Croes 1958; Mersmann et al. 2012), but if we use the same heuristic for other problem domains it might not perform that well. Thus a lot of effort has to be put into creating a good low level heuristic set.

To overcome this problem we propose a unified encoding. In this approach, the low level heuristics are not problem-specific, but are bound to the representation of the solution. Thanks to this approach, one can skip the difficult part of defining low level heuristics and get nonetheless good results. In the standard approach the domain barrier can be defined as a set of statistical measures gathered from the execution of low level heuristics and then used by the hyper-heuristic to pick one of them for the next iteration. The proposed unified encoding approach in Fig. 3 pushes the domain barrier one level down and can be interpreted as solution representation which is used by an evaluation function to calculate the objective value for a given solution.

The difference between the approaches can be expressed as follows. In the standard hyper-heuristic approach, given a set of problem-specific low-level heuristics and an objective function, one can solve problem P using one of the existing hyper-heuristics. In contrast, in the proposed approach: given a solution encoding and an objective function we can solve problem P using one of the existing hyper-heuristics and a generic set of solution-encoding-specific low-level heuristics.

In the evaluation of the framework, the solution is represented as a sequence S of unique integers from range 1 to n. Each instance is represented as a directed complete graph G with n vertices and weighted arcs between them. Data is stored in an $n \times n$ adjacency matrix D, where value $D_{i,j}$ denotes the weight of the arc from vertex i to vertex j. Graph G is a directed complete graph, so between every pair of vertices there are two arcs. Two real numbers are assigned to each vertex—the prize (Pr_i) and the penalty (Pe_i). There is also a constraint value represented as a real number (Q).

4.1 Problem domains and their respective representation

Four problem domains were analyzed when choosing the solution encoding and then a fifth problem domain was introduced to the framework as a hidden domain—it was not considered when the solution encoding was chosen. The problems are: the sequencing by hybridization problem (SBH), the TSP, the bottleneck traveling salesman problem (BTSP) and the prize collecting traveling salesman problem (PCTSP). The hidden domain was the knapsack problem.

4.2 Representing the sequencing by hybridization problem

Representing the SBH problem is done by setting the distances between vertices in matrix D as offset between l-mers and by setting constraint Q to the maximum length of the path. The values of Pe_i and Pr_i are set to 0 as they are not used.

The task is to maximize the following objective function:

$$|S|$$

Hard constraint:

$$\sum_{i=1}^{|S|-1} D_{S_i,S_{i+1}} \leq Q$$

This objective function does not distinguish between solutions of the same number of elements that differ in the route length. Keeping the hard constraint, an alternative objective function sbh-additive was proposed:

$$\text{sbh-additive} = (|S| + 1) \times factor - length$$

In this function, factor is a number that is significantly greater than the route length, and length is the length of the route expressed as the left side of the hard constraint equation. The scalar factor is used to leverage the main goal of visiting as many nodes as possible while keeping information about the differences in route length. To get the original objective value one has to take integral part of the fraction $\frac{sbh-additive}{factor}$.

4.3 Representing the traveling salesman problem

Mapping the TSP into the proposed model is trivial—matrix D is used to store the distances between cities as it would be in the original problem and the rest of the variables (Pr_i, Pe_i, Q) are equal to 0. If some edge in the original problem does not exist, it is represented as an edge with a very large weight which will not affect the correctness of the computed solution.

The task is to minimize the following objective function:

$$\sum_{i=1}^{|S|} D_{S_i, S_{(i \bmod n)+1}}$$

Hard constraint: The solution sequence S has to contain exactly n elements ($|S| = n$).

4.4 Representing the bottleneck traveling salesman problem

The BTSP instance is encoded in the same way as is done for the classic TSP—matrix D is used to store the distances between cities and Pr_i, Pe_i, Q are equal to 0. Missing edges are replaced with edges with a very large weight which will not affect the validity of the model.

The task is to minimize the following objective function:

$$\max_{i=1}^{|S|} D_{S_i, S_{(i \bmod n)+1}}$$

Hard constraint: The solution sequence S has to contain exactly n elements ($|S| = n$).

4.5 Representing the prize collecting traveling salesman problem

The PCTSP is the most complex problem in terms of input variables. The model actually perfectly fits with this problem and other problems can be easily transformed to this representation. In this problem, the task is to minimize the following objective function:

$$\sum_{i=1}^{|S|} D_{S_i, S_{(i \bmod |S|)+1}} + \sum_{v \notin S} Pe_v$$

Hard constraint:

$$\sum_{v \in S} Pr_v \geq Q$$

4.6 Representing the knapsack problem

To encode the knapsack problem with the proposed model the necessary values of item weights and item values must be mapped. An array of prizes Pr is used to store the values of items and array of penalties Pe is used to store their weights. Limit Q is used as the maximal total allowed weight of items in the knapsack. In this problem, matrix D is not used to evaluate the solution. Solution S will contain all items that are inside the knapsack.

The task is to maximize the following objective function:

$$\sum_{i=1}^{|S|} Pr_{S_i}$$

Hard constraint:

$$\sum_{i=1}^{|S|} Pe_{S_i} \leq Q$$

5 Low level heuristics

The low level heuristics are strictly bound to the solution encoding. They do not have any information about the problem domain, which makes them ineffective when used alone. The task of hyper-heuristics is to evaluate the performance of low level heuristics and select them according to relevant measures, similarly to the algorithm selection problem described previously.

Obviously, the set of low level heuristics should enable the search over the whole search space—starting from a given solution one should be able to reach another solution if applying certain moves to the starting solution. However, the low level heuristics might somehow overlap their behavior, because it is better to choose a single heuristics that is equivalent to a sequence of heuristics applied consecutively. A good example could be the move heuristic—it could be easily replaced by remove and insert heuristics applied one after another. However, in general, two heuristics need to be executed instead of one, which is considered to be inefficient.

In the following sections, five low level heuristics used for computational experiments are described: insert, remove, move, swap, replace. Those basic five heuristics can produce any possible solution but the set of low level heuristics can be easily extended. We performed an experiment with an extended set by introducing three more heuristics: move sequence, revert sequence and remove highest arc.

5.1 Insert heuristic

Input:
pos—a position in the sequence at which a node should be inserted
node—a node to be inserted

This heuristic basically tries to insert a new node into the sequence. By definition of the solution encoding, the *node* has to be unique, so it cannot be part of the sequence already. Thus, this heuristic is not useful for solutions of length n, because there is no unique node that can be inserted.

This heuristic had to be implemented in pair with the remove heuristic, so that every possible solution can be reached. However, for the TSP it is considered to be ineffective, because only solutions with all nodes are valid, for which there are no unique nodes to be inserted. Despite this fact, this heuristic can be used in the case of invalid starting solution of size smaller than n to repair the solution and reach the size of n for the TSP.

In the case of the PCTSP and the knapsack problem, the heuristic should play an important role, because there is no restriction on sequence length. For example, for a given capacity of the knapsack it might be possible to put another item into it, which will not produce an infeasible solution.

5.2 Remove heuristic

Input:
pos—a position in the sequence from which a node should be deleted

This heuristic is used to remove nodes from the solution to get shorter solutions. It is complementary with the insert heuristic—one can invert the effect of the other. With insert and remove heuristics it is possible to reach every possible solution from the search space. However, it might not be the fastest way to do so. Next three heuristics are in some way a combination of those two basic low level heuristics, but because they are considered as a single move, they might be more efficient.

The remove heuristics for the TSP will always produce infeasible solutions, because not all cities will be visited. Due to this fact, two heuristics that use a combination of insert and remove in a single operation are included, so the solutions will remain feasible. However, this heuristic can be useful for the prize collecting TSP and the knapsack problem. In the first one it can try to reduce the total route length but increase the punishment for paying penalties and for the latter it can, for example, change an infeasible, overweight solution into a feasible one or make space for other items.

5.3 Move heuristic

Input:
pos—a position in the sequence from which a node should be picked out
$dest$—a position in the sequence at which this node should be inserted

The move heuristic combines the insert and remove heuristics—it removes a node from pos and instantly inserts it at $dest$.

It was introduced for the case of the TSP and its bottleneck variant, where the solution must be of length n.

5.4 Swap heuristic

Input:
pos—a position in the sequence from which a node will be swapped
$pos2$—a second position in the sequence from which the other node will be swapped

This heuristics swaps two nodes which are at positions pos and $pos2$. It is similarly to the move heuristic in that it does not produce infeasible solutions for the TSP, because the length of the solution remains the same.

5.5 Replace heuristic

Input:
pos – a position in the sequence at which a node should be replaced
$node$ – a node to be inserted

This heuristic removes a node from position pos and inserts a $node$ which has not been in the solution, thus the sequence solution is unique. This heuristic was developed for the SBH and the PCTSP, because for other domains it will produce infeasible solutions. It should also be useful for the knapsack problem.

5.6 Move sequence heuristic

Input:
pos—a position in the sequence from which a sequence should be picked out
$length$—number of elements in the sequence
$dest$—a position in the sequence at which this nodes should be inserted

The move sequence heuristic is a generalisation of the move heuristic. It performs the same task but on a larger set of nodes. It was introduced for the case of the SBH, where the solution consists of many subsequences of good quality which are merged into a final sequence.

5.7 Revert heuristic

Input:
pos—a position in the sequence which is a first node in the subsequence
$pos2$—a position in the sequence which is the last node in the subsequence

This heuristic reverts the subsequence given by two nodes which are at positions pos and $pos2$. Assuming that $pos \leq pos2$, a node at position pos is swapped with node at position $pos2$, then node at position $pos + 1$ is swapped with node at position $pos2 - 1$, etc.

5.8 Remove highest arc in the solution heuristic

This heuristic looks for the highest arc in the sequence and replaces it with the one that improves the solution quality best. This is valid for all problems that use the matrix D to store a graph of distances between nodes. However, for the knapsack problem it has no effect and the solution is not changed. This heuristic is still problem independent, because it looks into the relation between the input data rather then the problem characteristics.

6 Computational experiments

The performance of the proposed approach can be analyzed from two points of view. First, a comparison to meta heuristic approaches for the SBH problem was made. Next, a comparison of different hyper-heuristic algorithms was investigated. The performance was measured as the distance of the result to the known optimal value expressed as a percentage.

6.1 Data set

To test any algorithm, a benchmark data set is required. Each problem is "similar" to another with respect to the used variables. However, their values are different and have different meanings. Thus, it is good practice to provide a set of instances crafted for a given problem domain. The used benchmark sets contain the known optimal value of the instances, which is beneficial for measuring the performance of algorithms.

- SBH—data used for SBH comes from (Blazewicz et al. 2006a). The optimal value is equal to the number of l-mers in instance minus positive errors count.
- TSPLib (inp, 1995)—this data set contains a large number of TSP instances. This benchmark was used for the classic TSP and BTSP. It provides optimum values for the first problem, but for reference on the second problem the instances from (Ramakrishnan et al. 2009; Larusic et al. 2012; Ahmed 2010) were used.
- UKP (inp, 2005)—this data set contains some knapsack instances with optimal values and the package also contains an instance generator.
- PCTSP—data used for this problem comes from (Chaves and Lorena 2008) in which a CPLEX method was used to get the optimal values.

All tests were conducted on a PC with Intel Xeon 2.33 GHz processor and 2,048 MB of physical memory, running openSUSE 11.4 operating system. The algorithms were implemented in Java, compiled and run in the JRE 6 Update 29 and were given a 5 min computation time.

6.2 Parameter tuning

The hyper-heuristic framework presented in this paper has a lot of parameters. The set of configurable input data consists of general, hyper-heuristic algorithm and low level heuristics parameters. The general category defines the stop condition, the hyper-heuristic algorithm and the set of low level heuristics which should be used to perform the search. As we mentioned in Sect. 6.1 we terminate the search after 5 min. All hyper-heuristics were tested in computational experiments and the set of low level heuristics contained all heuristics described in Sect. 5.

For each low level heuristic we can estimate the number of neighboring solutions by calculating the number of possible permutations. To search as many solutions as possible within the given amount of time we set the number of "inner iterations" (iterations made by low level heuristics itself to look for best neighborhood) to 5 % of all permutations or 100 iterations, whichever is smaller. With larger values we can get better local results, but this will take much more time.

To decide on parameters for hyper-heuristic algorithms, we performed initial computational experiment on small instances. The ranked-choice ranking length was set to 3 heuristics. Lower values make it similar to straight-choice and higher values often make one low-level heuristic superior above others. The tabu tenure was set to 4, because higher values would make it similar to round robin algorithm which leads to non-improving iterations. The initial temperature for simulated annealing was set to 0.015 with 13 iterations of cooling. The parameters α and β were set to 0.5, while γ was set to 10 and η was set to 0.1. Those parameters were picked in initial computational experiment with decomp-choice hyper-heuristics, because it is not dependent on any other factor.

6.3 Unified encoding applied to the SBH problem

We compare our proposed unified encoding approach with other methods dedicated to the SBH problem (Blazewicz et al. 2004, 2011). We used a set of instances from Blazewicz et al. (2006a). The dataset contains three different sizes of input set of l-mers (l was equal to 10). These l-mers were subfragments of original DNA sequences of length 200, 400 and 600, thus the size of the input sets was equal to 191, 391 and 591, respectively. The input set contained 5 % or 20 % error rate. 5 % error rate means that there are 5 % of negative errors (missing l-mers) and 5 % of positive errors (l-mers which do not occurred in the original sequence, but for some reason were detected during the biochemical experiment).

Table 1 contains the result of the computational experiments of all methods. We used the same simulated annealing hyper-heuristic as in Blazewicz et al. (2011), but provided it with our unified encoding and the set of low level heuristics. We also compared the ranked-choice hyper-heuristics. Two different measures were used to evaluate the algorithms. 'Avg. usage' is the percentage of l-mers from the spectrum used to construct the solution including only l-mers from original sequence, i.e. without positive errors. 'Alignment' is a measure used by biologists, because the most important thing for them is the similarity of the solution to the original sequence. It is calculated with the Needelman–Wunsch algorithm (Needleman and Wunsch 1970) after the computation, 100 % means that the sequences are identical.

The algorithms compared in the table have different levels of generalization. The tabu and scatter search (Blazewicz et al. 2004) is a meta heuristic algorithm tailored to solve the SBH problem. The simulated annealing hyper-heuristic from Blazewicz et al. (2011) is using a set of low level heuristics which are specially crafted for the SBH problem and it uses information about the problem domain—e.g. the algorithm gets better results by clustering the l-mers into longer sequences and then combining them together. With the proposed approach we use the same simulated annealing hyper-heuristic as a top level algorithm, but the set of low level heuristics is problem independent. The ranked-choice hyper-heuristic is also included in the comparison to present that it is capable of solving SBH problem. Those last two algorithms are using unified encoding and thus do not use any characteristics of problem domain.

Table 1 The results of the tests of different algorithms for biological dataset

Instance	200		400		600	
	5 %	20 %	5 %	20 %	5 %	20 %
Unified encoding + ranked-choice hyper-heuristics						
Avg. usage (%)	96,69	98.36	95.01	97.60	96.08	92.48
Alignment (%)	92.86	94.66	86.55	85.05	86.03	74.09
Unified encoding + simulated annealing hyper-heuristics						
Avg. usage (%)	97.79	96.38	97.17	98.88	98.84	92.27
Alignment (%)	94.63	89.95	90.47	90.58	93.84	73.33
Simulated annealing hyper-heuristics (Blazewicz et al. 2011)						
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
Tabu and scatter search meta heuristics (Blazewicz et al. 2004)						
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

Input datasets are created from DNA sequences which do not contain repetitions of l-mers. In the first row there is the length of the examined sequence, next is the percentage of errors in the spectrum. Each entry in rows 'Avg. usage' and 'Alignment' is the mean value across 40 instances

Obviously, with more generalization, one tends to obtain worse solutions. It is worth noting though, that the average usage of l-mers is comparable to the usage obtained by other algorithms, they differ by average of 2 %. In contrast, the alignment of the resulting sequences is not as good in comparison to other methods. However, it has to be stressed here that even for the ideal experiment it is possible to obtain different sequences from the same number of l-mers. With the increase of errors, the probability that for high usage the alignment is not satisfactory increases. There are a few instances for which the proposed new method allowed to obtain the original DNA sequence, i.e. the value of the alignment was equal to 100 %.

The proposed set of low level heuristics was very simple. It is highly possible that with more complex low level heuristics, which will still be problem independent, one could get even better results. The heuristics should have an impact on the search—either they should intensify the search by finding better combinations or diversify it by destroying good solutions.

6.4 Unified encoding performance for combinatorial problems

To compare different hyper-heuristics, we performed extensive tests on available benchmarking sets and calculated the distance to the optimal value (or reference value in the case of BTSP). For example, if the result is 25 % it means that the solution we obtained is 125 % of the optimum value, e.g. if the optimum is 100 then the obtained solution value is 125.

Figures 4 and 5 present the results of those experiments. The obtained performance for most domains is quite good if we take into account the amount of time spent on

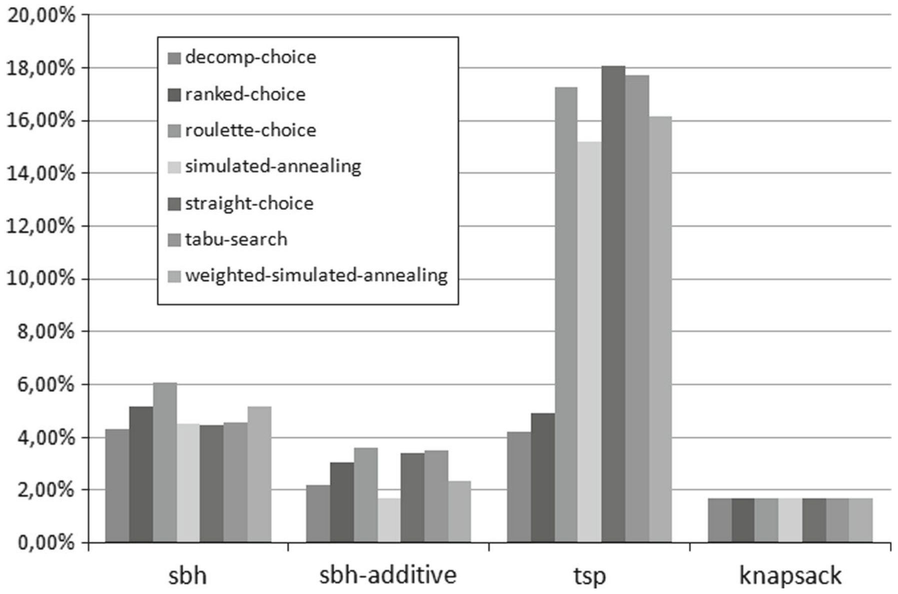


Fig. 4 Comparison of different hyper-heuristic algorithms. The *colors* present different hyper-heuristics. On the Y axis is the distance to the optimum value in %. Sbh-additive is enhanced objective function which allows to better differentiate solutions of the same number of l-mers

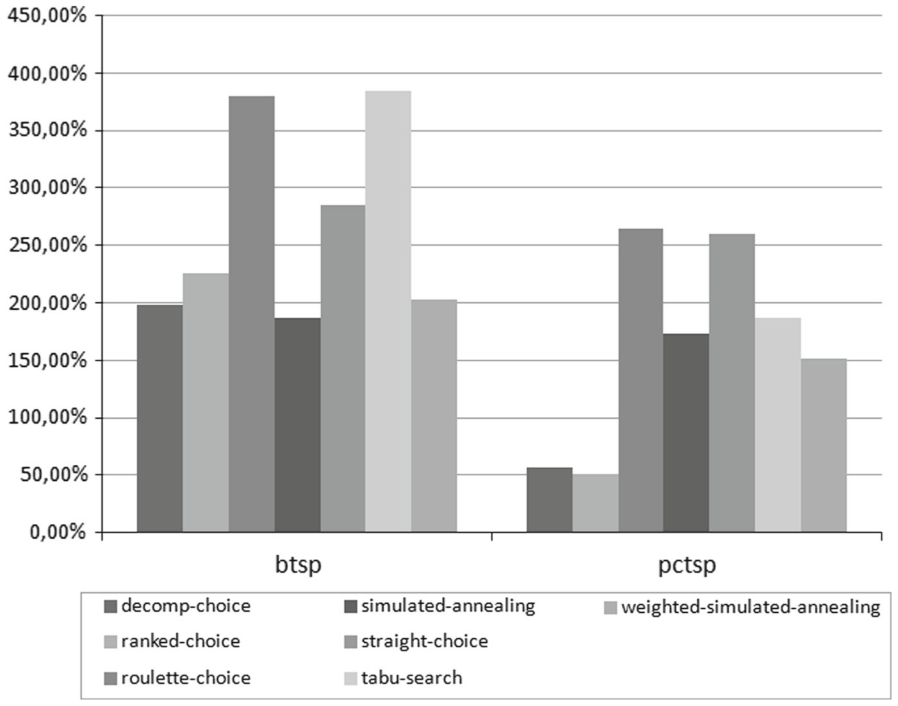


Fig. 5 Comparison of different hyper-heuristic algorithms. The *colors* present different hyper-heuristics. On the Y axis is the distance to the optimum value in %. For BTSP, a reference value was used

computing. In most cases the ranked-choice and decomp-choice hyper-heuristics performed best. For the knapsack problem, the SBH problem and the TSP, the proposed unified encoding used with one of those two hyper-heuristics gave results which differed on by at most 5 % from the optimum, even though these problems are of different type.

On the other hand, it did not cope well with the BTSP and the PCTSP. The first problem is particularly hard, because the objective value does not change continuously and there are many solutions with the same value which are hard to distinguish. There is potential in overcoming this issue by providing an objective function that will distinguish the solutions in a better fashion.

In contrast, the PCTSP is much more diverse, but the set of low level heuristics was not enough to solve it effectively. However, the ranked-choice and decomp-choice hyper-heuristics clearly reduced the usage of ineffective low level heuristics from the set and obtained better results then other algorithms. This leads to a conclusion that even though the low level heuristics are problem independent, they have significant influence on the search effectiveness.

7 Conclusions

In this paper, we have proposed a unified encoding for the hyper-heuristic approach, which should provide an easier way to address different combinatorial problems. We introduced more generalization (problem independency) by employing the unified encoding and using a set of low level heuristics which are specific to that encoding rather than to the particular problem domain. The off-the-peg solution that will simplify the solution procedure even when compared to a standard hyper-heuristic approach is desirable, because one does not need to analyze the problem characteristics to create sophisticated algorithms or low level heuristics and yet we can still generate approximate solutions that are good enough. To solve a new problem the user has to define the objective function which will be optimized and eventually change the input parameters of the methods used within the framework.

However, the set of low heuristics could be extended to exploit more complex operations, although the basic set provided for the testing purposes gave surprisingly good results, especially when we introduced the move sequence, revert sequence and remove arc heuristics. From the tested hyper-heuristics, the simulated annealing hyper-heuristic and ranked-choice hyper-heuristic are the most promising as they enable intensification and diversification of the search procedure by using low level heuristics which decreases the objective value for different problem domains. When compared to some meta heuristic implementations and standard hyper-heuristics, the proposed unified encoding performed well, but by introducing more low level heuristics (to get better coverage of the solution space) it is possible to obtain even better results. It would be interesting to investigate both the behavior and the usage of the low level heuristics bound to the unified encoding for different problem domains and to define characteristics of those problems to better understand their nature. Also, it could be interesting to investigate other hyper-heuristics and see if any of them gives significantly better results.

Acknowledgments This research project was partially supported by Grant no. DEC-2011/01/B/ST6/07021 and Grant no. DEC-2012/05/B/ST6/03026 from the National Science Centre, Poland.

References

Aarts E, Korst J, Michiels W (2005) Simulated annealing. In: search methodologies: introductory tutorials in optimization and decision support, techniques, pp 187–210

Ahmed ZH (2010) A lexisearch algorithm for the bottleneck traveling salesman problem. Int J Comput Sci Secur 3(6):569–577

Bai R, Kendall G (2005) An investigation of automated planograms using a simulated annealing based hyper-heuristic. In: Metaheuristics: progress as real problem solvers operations research/computer science interfaces series, vol 32, pp 87–108

Blazewicz J, Kaczmarek J, Kasprzak M, Markiewicz W, Weglarz J (1997) Sequential and parallel algorithms for DNA sequencing. CABiOS 13:151–158

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

Blazewicz J, Kasprzak M (2003) Complexity of DNA sequencing by hybrydization. Theor Comput Sci 290:1459–1473

Blazewicz J, Glover F, Kasprzak M (2004) DNA sequencing—tabu and scatter search combined. INFORMS J Comput 16(3):232–240

Blazewicz J, Glover F, Swiercz A, Kasprzak M, Markiewicz W, Oguz C, Rebholz-Schuhmann D (2006a) Dealing with repetitions in sequencing by hybridization. Comput Biol Chem 30(5):313–320

Blazewicz J, Oguz C, Swiercz A, Weglarz J (2006b) DNA sequencing by hybridization via genetic search. Oper Res 54:1185–1192

Blazewicz J, Burke EK, Kendall G, Mruczkiewicz W, Oguz C, Swiercz A (2011) A hyper-heuristic approach to sequencing by hybridization of DNA sequences. Ann Oper Res, pp 1–15. doi:10.1007/s10479-011-0927-y

Bui T, Youssef W (2004) An enhanced genetic algorithm for DNA sequencing by hybrydization with positive and negative errors. Lect Notes Comput Sci 3103:908–919

Burke EK, Gendreau M, Hyde M, Kendall G, Ochoa G, Özcan E, Qu R (2013) Hyper-heuristics: a survey of the state of the art. Journal of the Operational Research Society (10 July 2013), Palgrave Macmillan

Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E, Woodward JR (2010) A classification of hyper-heuristic approaches. In: Gendreau M, Potvin JY (eds) Handbook of meta-heuristics, vol 146. Springer, International series in operations research and management science, pp 449–468

Burke EK, Kendall G, Soubeiga E (2003b) A tabu-search hyperheuristic for timetabling and rostering. J Heuristics 9:451–470

Burke EK, McCollum B, Meisels A, Petrovic S, Qu R (2007) A graph-based hyper-heuristic for timetabling problems. Eur J Oper Res 176:177–192

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, international series in operations research and management science, vol 57. Springer, New York, chap 16, pp 457–474. doi:10.1007/0-306-48056-5_16

Chaves AA, Lorena LAN (2008) Hybrid metaheuristic for the prize collecting travelling salesman problem. In: Proceedings of the 8th European conference on evolutionary computation in combinatorial optimization. Springer-Verlag, Berlin, Heidelberg, EvoCOP'08, pp 123–134. http://dl.acm.org/citation.cfm?id=1792634.1792645

Cichowicz T, Drozdowski M, Frankiewicz M, Pawlak G, Rytwińnski F, Wasilewski J (2012) Five phase and genetic hive hyper-heuristics for the cross-domain search. In: Lecture notes in computer science 7219. Springer 2012, pp 354–359

Cichowicz T, Drozdowski M, Frankiewicz M, Pawlak G, Rytwińnski F, Wasilewski J (2012) Hyper-heuristics for cross-domain search. In: Bulletion of the Polish Academy of Sciences. Technical Sciences 60(4):801–808

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. Springer-Verlag, London, UK, vol 2079, pp 176–190. http://portal.acm.org/citation.cfm?id=646431.692903

Cowling P, Kendall G, Soubeiga E (2002a) Choice function and random hyperheuristics. In: Proceedings of the fourth Asia-Pacific conference on simulated evolution and learning, SEAL, Springer, pp 667–671

Cowling P, Kendall G, Soubeiga E (2002b) Hyperheuristics: a tool for rapid prototyping in scheduling and optimisation. In: Proceedings of the applications of evolutionary computing on EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN, Springer-Verlag, London, UK, pp 1–10. http://dl.acm.org/citation.cfm?id=645407.652005

Croes GA (1958) A method for solving traveling-salesman problem. In: Operations Research. INFORMS 6:791–812

Dowsland KA (1998) Off-the-peg or made-to-measure? Timetabling and scheduling with SA and TS. In: Selected papers from the second international conference on practice and theory of automated timetabling II. Springer-Verlag, London, UK, PATAT '97, pp 37–52. http://dl.acm.org/citation.cfm?id=646430.692896

Dowsland K, Soubeiga E, Burke EK (2007) A simulated annealing hyper-heuristic for determining shipper sizes. Eur J Oper Res 179:759–774

Dramanac R, Labat I, Brukner I, Crkvenjakov R (1989) Sequencing of megabase plus DNA by hybrydization: theory of the method. Genomics 4:114–128

Gendrau M, Potvin JY (2005) Tabu search. In: Search methodologies: introductory tutorials in optimization and decision support, techniques, pp 165–186

Glover F, Laguna M (1997) Tabu search. Kluwer Academic Publishers, Norwell

Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW (eds) Proceedings of a symposium on the complexity of computer computations, the IBM research symposia series. Plenum Press, New York, pp 85–103

Kendall G, Hussin NM (2005) A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA University of Technology. Lect Notes Comput Sci 3616:270–293

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science, Number 4598(13), May 1983, 20, 4598:671–680. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.4175

Larusic J, Punnen AP, Aubanel E (2012) Experimental analysis of heuristics for the bottleneck traveling salesman problem. J Heuristics 18(3):473–503

Lysov LP, Florent'ev VL, Khorlin AA, Khrapko KR, Shik VV (1988) Determination of the nucleotide sequence of DNA using hybridization with oligonucleotides. A new method. In: Doklady Akademii nauk SSSR, vol 303:1508–1511

Mersmann O, Bischl B, Bossek J, Trautmann H, Wagner M, Neumann F (2012) Local search and the traveling salesman problem: a feature-based characterization of problem hardness. In: Learning and Intelligent Optimization conference, Microsoft Technology Center, Paris

Mruczkiewicz W (2009) Hyper-heuristics for Sequencing by Hybridisation Problem. Master's thesis, Poznan University of Technology, Poznan, Poland

Needleman SB, Wunsch CD (1970) A general method applicable to search for similarities of the amino acid sequence of two proteins. J Mol Biol 48:443–453

Özcan E, Bilgin B, Korkmaz EE (2008) A comprehensive analysis of hyper-heuristics. Intell Data Anal 12(1):3–23. http://dl.acm.org/citation.cfm?id=1368027.1368029

Pevzner PA (1989) l-tuple DNA sequencing: computer analysis. J Biomol Struct Dyn 7(1):63–73

Ramakrishnan R, Sharma P, Punnen A (2009) An efficient heuristic algorithm for the bottleneck traveling salesman problem. OPSEARCH 46:275–288. doi:10.1007/s12597-009-0018-x

Rice JR (1976) The algorithm selection problem. In: Rubinoff M, Yovits MC (eds) Advances in computers, vol 15, Elsevier, pp 65–118. doi:10.1016/S0065-2458(08)60520-3

Ross P (2005) Hyper-heuristics. In: Search methodologies: introductory tutorials in optimization and decision support, techniques, pp 529–556

Ross P, Marin-Blázques JG, 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

Smith-Miles K, Lopes L (2012) Measuring instance difficulty for combinatorial optimization problems. Comput Oper Res 39(5):875–889. doi:10.1016/j.cor.2011.07.006

Southern E (1988) United Kingdom Patent Application GB8810400
TSP Lib Benchmark (1995) [on-line] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
UKP Knapsack Benchmark (2005) [on-line] http://download.gna.org/pyasukp/