Hybrid Metaheuristics: An Automated Approach

Ahmed Hassan[a,*], Nelishia Pillay[a,b]

[a] *University of KwaZulu-Natal, Pietermaritzburg 3201, South Africa*
[b] *University of Pretoria, Pretoria 0002, South Africa*

Abstract

Hybrid metaheuristics have proven to be effective at solving complex real-world problems. However, designing hybrid metaheuristics is extremely time consuming and requires expert knowledge of the different metaheuristics that are hybridized. In previous work, the effectiveness of automating the design of relay hybrid metaheuristics has been established. A genetic algorithm was used to determine the sequence of hybridized metaheuristics and the parameters of the metaheuristics in the hybrid. This study extends this idea by automating the design of each metaheuristic involved in the hybridization in addition to automating the design of the hybridization. A template is specified for each metaheuristic, defining the metaheuristic in terms of components. Manual design of metaheuristics usually involves determining the components of the metaheuristic. In this study, a genetic algorithm is employed to determine the components and parameters for each metaheuristic as well as the sequence of hybridized metaheuristics. The proposed genetic algorithm approach was evaluated by using it to automatically design hybrid metaheuristics for two problem domains, namely, the aircraft landing problem and the two-dimensional bin packing problem. The automatically designed hybrid metaheuristics were found to perform competitively to state-of-the-art hybridized metaheuristics for both problems. Future research will extend these ideas by looking at automating the derivation of metaheuristic algorithms without predefined structures specified by the

*Corresponding author
Email addresses: ahmedhassan@aims.ac.za (Ahmed Hassan), npillay@cs.up.ac.za & pillayn32@ukzn.ac.za (Nelishia Pillay)

templates.

Keywords: hybrid metaheuristic, meta-genetic algorithm, automated design

---

Declarations of Interest

Declarations of interest: none

1. Introduction

As we move into the fourth industrial revolution the automated design of machine learning and search techniques is a rapidly growing field. There has been a fair amount of research into the automated design of metaheuristics, with metaheuristics often being used to design metaheuristics which has led to the field of multilevel metaheuristics Sevaux et al. (2018). Tabli Talbi (2009) refer to metaheuristics that optimize the parameters of other metaheuristics as specialist high level relay hybrid metaheuristics. Design decisions include deciding on which parameters and operators to use, the control flow of algorithms, the derivation of new operators, determining which methods to hybridize and how to hybridize them. These design decisions are time-consuming, requiring many man-hours and expert knowledge. Automated design of machine learning and search techniques aims to alleviate researchers and practitioners from the laborious and time-consuming task of design, also enabling practitioners without expert knowledge to rather focus on the application at hand Nyathi & Pillay (2018).

In previous work the effectiveness of automating the design of relay hybrid metaheuristics has been established Hassan & Pillay (2017, 2018). In these studies a genetic algorithm has been used to determine which metaheuristics to hybridize; the sequence in which to apply the metaheuristics and the parameter values for each metaheuristic in the hybridization. In the later study, the effectiveness of the hybrid metaheuristics produced by the automated design over the individual metaheuristics tuned using IRACE was illustrated. The study presented in this paper extends this work by automating the design of each individual metaheuristic in addition to automating the hybridization of the metaheuristics and selection of parameters simultaneously.

The hybrids that are produced by the genetic algorithm are high level relay (HRH) metaheuristics Talbi (2009). Both S-metaheuristics and P-metaheuristics are combined, these are referred to as single-point and multipoint search respectively in the paper. The hybrid metaheuristics produced are heterogeneous hybrids as they are composed of different metaheuristics and the hybrids are global, general hybrids as they explore the space to solve a particular optimization problem.

A template is defined for each metaheuristic, composed of components corresponding to design decisions, e.g. which operator to use. A genetic algorithm is used to determine the components and parameters for each metaheuristic, the metaheuristics to hybridize and the order in which the metaheuristics are applied in the hybrid. The proposed approach for the automated design of hybrid metaheuristics has been evaluated on two real-world problems, namely, the aircraft landing problem and the two-dimensional bin packing problem. In both these instances, the automated designed hybrid metaheuristics were found to perform better than manually designed hybrid metaheuristics, producing results competitive to the state of the art techniques. Hence, the main contribution of the research presented in this paper is the automated design of the individual metaheuristics in the hybridization in addition to the hybridization of the metaheuristics. To the knowledge of the authors, this has not been previously investigated. Please note that the scope of the research is restricted to the application of a metaheuristic, in this case a genetic algorithm, to combine metaheuristics in sequence to produce a hybrid and does not include other mechanisms for combining metaheuristics such as portfolios of metaheuristics.

The main aim of the study is to determine whether automating the design of the metaheuristics and hybridization of the metaheuristic simultaneously produces hybrid metaheuristics that perform just as well as the manually designed hybrid metaheuristics. Please note that the aim of the study is not to determine how the performance of the hybridized metaheuristics compare to the individual metaheuristics, previous work in the field has already ascertained that hybridized metaheuristics perform better than individual metaheuristics for the particular domains investigated in the study Hassan & Pillay (2018). Furthermore, the study does not aim to ascertain that individual metaheuristics produced by automated design perform as well as manually design metaheuristics, the effectiveness of the automated design of individual metaheuristics has also been established by numerous previous studies. The study is aimed at determining whether automating the design of the individual metaheuristics and hybridization of the metaheuristics simultaneously is at least just as effective as the manual design of hybridizing the metaheuristics and the individual heuristics in the hybridization. Hence the performance of the hybrid metaheuristics produced by the automated design is compared to that of manually designed hybrid metaheuristics for the same problems. The individual metaheuristics in the latter are also manually designed.

The rest of the paper is structured as follows. In Section 2, a review of related work is presented. The proposed automated approach is described in Section 3. Section 4 presents the experimental setup. In Section 5, the results are discussed. Section 6 concludes the paper.

2. Related Work

The autonomy of AI methods in general and search methods in particular has been addressed by the AI community using various approaches including, but not limited to, automatic algorithm configuration Hutter et al. (2007); López-Ibánez et al. (2011), algorithm selection Rice (1976), algorithm portfolio Gomes & Selman (2001), reactive search Battiti et al. (2008) and hyper-heuristics Burke et al. (2013). Fuzzy logic has also been successfully applied for automated design including parameter tuning in metaheuristics Olivas et al. (2019); Valdez et al. (2014) and the design of neural networks González et al. (2015).

As explained in Section 1, the research presented in this paper extends previous work by automating the design of individual metaheuristics in terms of the components of the metaheuristics and parameters as well as automating the design of the hybridization of the individual metaheuristics. Hence, this section provides an overview of previous studies conducted to automate the design of metaheuristics with respect to deciding on the components of the metaheuristic algorithm, the parameter values for the metaheuristic and the combination of metaheuristics to solve the problem at hand.

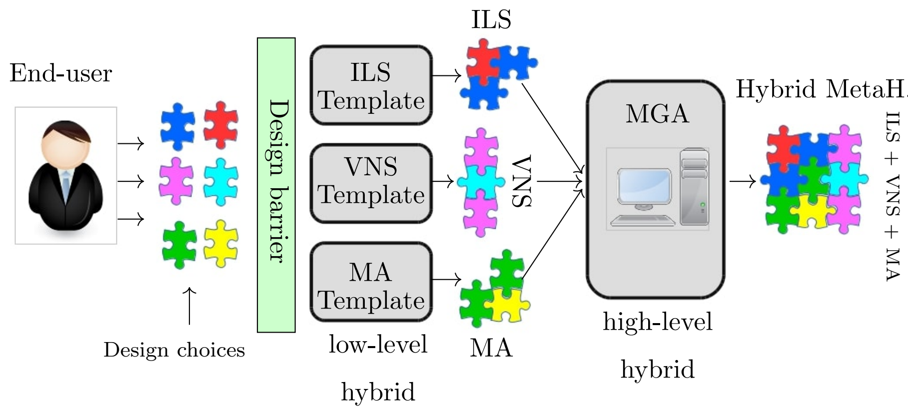
Oltean (2005) proposes linear genetic programming (GP) to evolve evolutionary algorithms (EA) using alternative genetic operators. Tavares & Pereira (2012) use GE to design ant colony algorithms (ACO) out of existing components of ACO such as solution construction, reinforcement and evaporation. Tavares & Pereira (2011) use a GP algorithm to design a strategy for pheromone update in ACO. Dioşan & Oltean (2009) use a meta-genetic algorithm (MGA) to automate the discovery of optimal structures and parameters for EA where each individual of MGA is an EA. Drake et al. (2013) use a GE hyper-heuristic to automate two components of variable neighborhood search namely, solution construction heuristics and neighborhood operators. Keller & Poli (2007) design basic metaheuristics using a linear GP hyper-heuristic relying on a simple grammar that specifies general primitives such as loops and conditional statements as well as problem-specific primitives such as the 2-OPT and 3-OPT operators for the TSP. Sabar et al. (2013) use a GE hyper-heuristic that evolves local search heuristics from three basic components which are move acceptance criteria, neighborhood structures and neighborhood combinations. Bhanu & Gopalan (2008) propose a hyper-heuristic with a greedy selection strategy to select the best performing metaheuristic from among a genetic algorithm and three hybrid genetic algorithms. Grobler et al. (2010) propose a selection hyper-heuristic working on a set of low-level heuristics containing EA, differential evolution and particle swarm optimization algorithms. That work is extended by Grobler et al. (2012) through incorporating local search procedures. Garcia-Villoria et al. (2011) present a hyper-heuristic consisting of a learning phase during which single-point searches are used and a launching phase during which the best-performing search during the learning phase is selected and applied to the best solution. Tsai et al. (2012) use a simple random hyper-heuristic to hybridize tabu search, simulated annealing, k-means algorithm and a genetic algorithm. Hassan & Pillay (2017) propose a hyper-heuristic-like MGA to automatically decides the best sequence of metaheuristics to use as well as the parameter values of the metaheuristics in the sequence. Although not particularly related to our work (as defined above), it is worth mentioning that automatic algorithm configuration tools (AACT) are used in the context of the automated design of metaheuristics/hybrid metaheuristics; for instance Marmion et al. (2013); López-Ibánez et al. (2017) use AACT to automatically design stochastic local searches and Lopez-Ibanez & Stutzle (2012) use AACT to design multi-objective ACO algorithms. In this case, instead of the traditional use of AACT to tune the parameter of a well-defined algorithm, it is actually used to design metaheuristics/hybrid metaheuristics (together with parameter tuning) through expressing metaheuristics via a parameteric representation.

The study presented in this paper focuses on the automated design of relay hybrid metaheuristics. Please note that our proposed approach is distinct from prior related work in a number of major aspects. *a*) In this study, each metaheuristic in the sequence is automatically designed and the best way by which metaheuristics can be sequenced is also automatically decided. *b*) In this study, single-point and multi-point metaheuristics are hybridized within a single framework which is a lacking feature in most of the prior studies. *c*) This study automates the design of hybrid metaheuristics unlike some of the prior studies which automate the design of a specific metaheuristic such as ant colony optimization algorithms. *d*) Our proposed approach caters for automatic parameter tuning as part of the design process without relying on a third-party software to perform this task.

3. Proposed Method

A steady-state meta-genetic algorithm MGA is used to automate the design of each metaheuristic in terms of selecting components for the defined templates and parameter values as well as the hybridization of the metaheuristics to solve the problem at hand. The proposed approach is illustrated visually in Figure 1. The end user provides the design choices for the predefined algorithmic templates; however, he/she is not responsible for carrying out the design manually; hence, there is a design barrier in the figure to emphasize this aspect. The provided design choices are used to create metaheuristics from the templates during the chromosome initialization as will be explained in Section 3.1. The created metaheuristics are combined sequentially and evolved using the MGA. The MGA is explained in detail in following subsections.

Figure 1: Framework of the proposed approach.



3.1. Initial Population and Chromosomes

The initial population is comprised of chromosomes of variable length ranging from 1 to L where the minimum length is set to 1 to verify whether the sequential hybridization is actually needed and the maximum length is set to L to prevent too long chromosomes from being created in the first generation. For later generations, longer chromosomes will not be created either as a result of using an aggressive crossover operator that causes the offspring to inherit the length as well as the genetic materials from the parents (see Section 3.3). Duplicates are not allowed in the initial population. Two chromosomes are considered duplicates if and only if they have the same length and the corresponding genes in both chromosomes encode the same metaheuristics. Two metaheuristics are the same if and only if they are instantiated from the same template using the same design choices and the same parameter values. Therefore, one difference in one parameter value can distinguish metaheuristics. This makes sense as the performance of metaheuristics is sensitive to changing their parameter values.

A chromosome of length l is a sequence $g_1 g_2 \dots g_l$ of l genes where each gene encodes a metaheuristic. A gene g_i is expressed in the form $h_i : s_i$ where h_i designates one of the templates and s_i is the specification needed to create a functional metaheuristic from h_i. In particular, s_i specifies the particular design choices and parameter values for the components of h_i. To illustrate, let h_i refer to the template of iterated local search (ILS) described in Algorithm 1 below. The template consists of three components: `localSearch`, `perturb` and `acceptCriterion`. Therefore, s_i should determine a specific design choice for each one of these three components in order to have an operational ILS algorithm. For example, the `localSearch` component could be assigned a simple hill-climbing local search, the `perturb` component could be assigned a random perturbation operator and the `acceptCriterion` component could be assigned `accept-all-moves`.

During the chromosome construction, each gene is created at random as follows. First, h_i is chosen at random from the three predefined algorithmic templates described in algorithms 1, 2 and 3 below. Then, for each component of h_i, a design choice is chosen at random from a predefined set of available design choices for that component (see Appendix A for a complete list of the design choices). When a design choice depends on parameters, a specific value for each parameter is chosen at random from a predefined range. The specification s_i is passed to the template h_i and decoded to instantiate an operational metaheuristic. Although the initial population is comprised of chromosomes that are created purely at random without an intelligent method, chromosomes that encode poor design choices and/or bad parameter values will be eliminated as the MGA evolves the population over generations.

Algorithmic Templates. As mentioned earlier in this section, each gene in the chromosome encodes a single metaheuristic created from a predefined algorithmic template by assigning a specific design choice for each component of the template. There are three algorithmic templates used. The first template, presented in Algorithm 1, describes iterated local search (ILS) generally without reference to a particular instance of ILS.

Algorithm 1 General Template of Iterated Local Search

Require: Initial Solution S_0

```
1: S* ← localSearch(S_0)
2: repeat
3:     S' ← perturb(S*, ρ)
4:     S'' ← localSearch(S')
5:     S* ← acceptCriterion(S*, S'')
6: until Stopping condition is met
```

The routine `pertrub` generates a new solution S' from the current local opitmum S^* by altering some components of S^*; the routine `localSearch` performs local search on S' to generate a new local optimum S'' which overwrites S^* (accepted) according to the routine `acceptCriterion`. Please see Gendreau & Potvin (2010) for more details about ILS.

The second template is for variable neighborhood search (VNS) which is specified by Algorithm 2.

The routine `shake` generates a new solution S' from the current solution S using the k^{th} neighborhood structure (NS). The routine `localSearch` performs local search on S' to generate a local optimum S''. The routine `changeNeigh` changes NS systematically using the feedback provided by S, S'' and k. Please see Hansen et al. (2010) for further details about VNS.

The last template is for memetic algorithms (MA) which is described by Algorithm 3.

The routine `select` chooses parents for mating. The crossover and mutation are done by the routine `regenerate` to produce offspring from the selected parents. The routine `localSearch` performs local search on the generated off-

Algorithm 2 General Template of Variable Neighborhood Search

Require: Initial Solution S

```
repeat
    k ← 1
    repeat
        S' ← shake(S, k)
        S'' ← localSearch(S')
        keepBest(S, S'')
        changeNeigh(S, S'', k)
    until k ← k_max
until Stopping condition is met
```

Algorithm 3 General Template Memetic Algorithm

```
initialize the first population
evaluate each individual
repeat
    matingPool ← select()
    offspring ← regenerate(matingPool)
    localSearch(offspring)
    reproduce(offspring)
until stopping condition is met
```

spring. The routine `reproduce` replaces the worst individual by the offspring if it is better. Please see Moscato et al. (1989) for more details about MAs.

3.2. Fitness and Selection

Each chromosome is evaluated by using the metaheuristics encoded by the genes of the chromosome to solve the training instances. As the aim of the study is to produce a hybrid that will work well over a set of problem instances, a training set is used instead of a single problem instance for evaluation. A naive fitness measure is the average objective value over the training instances. However, such a fitness measure is susceptible to exceptionally good/bad performance in specific instances. In this study, the fitness of a chromosome is defined relatively as done in Hassan & Pillay (2018).

$$F_i = 1 + \sum_{i=1}^{P} \delta_{ij}, \quad j = 1, \ldots, P, \quad \text{and} \quad j \neq i$$

where P is the population size and the stepwise function δ_{ij} is one if the number of instances in which chromosome i performs strictly better than chromosome j is larger than the number of instances in which chromosome j performs no worse than chromosome i and is zero otherwise. This fitness favors chromosomes with good performance on the majority of the training instances.

To make the fitness evaluation meaningful, all chromosomes use the same training instances and start from the same initial solutions constructed using problem-specific procedures described in Appendix B. In addition, all chromosomes are allocated the same computational budget (number of iterations) to prevent longer chromosomes from overtaking shorter ones just because of using a more computational budget. If a chromosome consists of more than one gene, the total computational budget is divided equally among the genes.

Tournament selection is used to choose parents for mating to produce the individuals of the next generation. From the current population, t individuals are chosen uniformly at random where t is the tournament size. Then, a tournament is run and the individual with the best fitness wins the tournament, i.e. is selected as a parent to produce offspring.

3.3. Regeneration

The design of hybrid metaheuristics is a time-consuming task such that it is crucial to use crossover operators that accelerate the evolution. To this end, an aggressive crossover is used which is tailored towards eliminating bad building blocks. In this context, bad building blocks refer to genes representing metaheuristics that perform poorly due to either bad design choices or bad parameter values. The crossover used in this study is an extension of the fitness-based scanning crossover (FBS) Eiben et al. (1994). FBS is a multi-parent crossover operator in which genes of the offspring are selected probabilistically from the parents in the mating pool where the probability of selecting a gene from a particular parent is proportional to its fitness. The extension of FBS covers two aspects. First, FBS is extended to work on individuals of variable length by setting the length of the offspring equal to the length of the best parent in the mating pool. Second, the number of parents in the mating pool is made a uniform random variable as apposed to being a predetermined fixed number. As a consequence of having parents of variable length, it is necessary to update the probability distribution whenever the offspring becomes longer than at least one parent because such a parent can no longer contribute to the subsequent genes of the offspring.

The generated offspring is mutated at a single gene chosen at random by replacing the chosen gene with a randomly created gene. The specific details on the random creation of a gene is mentioned above under Initial Population. The offspring replaces the worst individual if it is better.

3.4. Interaction Scheme

The communication between the genes of a chromosome enables later genes to benefit from the effort of earlier genes. In general, the best solution found by a preceding gene is passed to the subsequent gene as an initial solution except when the subsequent gene is a population-based method in which case all the best solutions found by the previous genes are embedded in the initial population and the rest of the population is created at random. For the first gene, the initial solutions are created using problem-specific solution construction methods; see Appendix B.

4. Experimental Setup

In this study, the *design phase* is separated from the *application phase*. In the design phase, the proposed approach uses the training instances to automatically design a high-performing hybrid metaheuristic. In the application phase, the automatically designed hybrid solver (ADHS) is evaluated by solving the test set instances. The automated approach plays the role of the human designer and the ADHS is the end product of the automated design process which corresponds to the manually designed published methods. Therefore, the ADHS is compared with the published methods; not the automated approach itself.

4.1. Problem Domains

The aircraft landing problem and the two-dimensional bin packing problem are used to evaluate the ADHS generated by the proposed automated approach. These problems are chosen since they hard to solve, have practical relevance and have been subjects of research for a few decades; consequently, the existing state-of-the-art approaches are advanced and hard to beat unless the proposed automated solvers are well designed.

4.1.1. Aircraft Landing Problem

Aircraft Landing Problem (ALP) is an NP- hard problem Beasley et al. (2000) that contains two subproblems which are sequencing and scheduling Ernst et al. (1999). ALP is hard to solve exactly even for medium-size instances Salehipour et al. (2013). The task of ALP is allocating runways and assigning landing times for arriving planes at an airport such that a certain objective, such as air traffic congestion, is optimized whereas each plane's landing time falls within a time window and a minimum separation criterion (also known as the safety constraint Sabar & Kendall (2015)) between every pair of planes is not violated. Due to its practical relevance, various models and methods are proposed for the ALP Girish (2016). There are two variants of ALP in the literature: the static case and the dynamic case. In the static case Beasley et al. (2000), complete information about all planes in the planning horizon is known a priori. In the dynamic case Beasley et al. (2004), new information is obtained as new arriving planes are picked by the airport's radar. In this study, the static case is considered.

4.1.2. Two-Dimensional Bin Packing Problem

The two-dimensional bin packing problem (2BPP) is a well-known NP-hard problem Lodi et al. (2002a,b). The classical one-dimensional bin packing problem is a special case of 2BPP Lodi et al. (2002b). 2BPP is concerned with packing a finite number of 2D items into 2D bins such that the total number of used bins is minimized Lodi et al. (2002a,b). In this study, the items and the bins are assumed to be rectangular and bins are identical. The packing is orthogonal, i.e, the sides of the packed items are either parallel or perpendicular to the sides of the bin. In the literature Lodi et al. (1999b), there are four variants of the orthogonal, rectangular 2BPP depending on whether the items have fixed orientation and whether the guillotine constraint is imposed. The variant considered in this study is well investigated by prior research and is known as 2BP|O|F where O indicates that the items are oriented and F indicates that the cutting is free, i.e. the guillotine constraint is not imposed. 2BPP has practical applications in many industries especially those involving mass-production where small savings on the raw material leads to a high reduction in cost Hopper (2000). Different mathematical models are proposed for the 2BPP; see Lodi et al. (2002a).

4.2. Benchmark Datasets

ALP. The benchmark dataset of ALP is proposed in Beasley et al. (2000, 2004) and is downloadable from OR-LIBRARY.[1] The dataset contains 49 instances ranging from 10 to 500 planes with a single runway to five multiple runways.

2BPP. The benchmark dataset of the 2BPP is proposed by Berkey & Wang (1987); Martello & Vigo (1998) and is downloadable from BPPLIB.[2] The dataset is grouped into 10 classes. Each class is subdivided into 5 categories. Each category contains 10 instances of sizes 20, 40, 60, 80 or 100. In total, there are 500 instances.

[1] http://people.brunel.ac.uk/ mastjjb/jeb/orlib/airlandinfo.html.

[2] http://or.dei.unibo.it/library/two-dimensional-bin-packing-problem

In both problems, the dataset is divided into two subsets: the *training set* which is used during the design phase to train the MGA to evolve high-quality hybrid metaheuristics and the *test set* which is used during the application phase to evaluate the effectiveness of the evolved hybrid metaheuristics. In both problems, the training set instances are chosen at random such that the training set size is 10% of the size of the entire dataset. For the ALP, the training set is chosen as follows. First the entire dataset is divided into two subsets: the easy instances (with sizes less than or equal 50) and the hard instances (with sizes greater than 50). One instance from the easy instances is chosen at random and its number of runways is also chosen at random. The same process is repeated for selecting training instances from the set of hard instances. For the 2BPP, one instance is chosen at random from each category.

4.3. Implementation Platform

As mentioned earlier in this section, the simulation conducted in this study is of two phases: the design phase during which hybrid metaheuristics are automatically designed using the MGA and the application phase during which the automatically designed hybrid metaheuristic is used to solve the test set instances. The design phase is run on a cluster having CPUs (2.6 GHz) and interconnected with FDR 56 GHz InfiniBand with CentOS 7.0. The application phase is run on a desktop computer with Intel Core i7 processor (3.10GHz) and 7.7 GiB of memory with Ubuntu 16.04 (64-bit). The programming language is Java 8 in both cases.

5. Result and Discussion

The MGA is run until convergence which is signaled by the absence of improvement for a number of generations. The best evolved algorithm (henceforth, referred to as automatically designed hybrid solver (ADHS)) is used to solve the test set instances. Then, the results of ADHS are compared with those of the published methods. As one of the design decisions that have to be made is which metaheuristics will be included in the hybrid and different designers use different metaheuristics we compare the performance of the hybrid produced by the automated design to more than one manually designed hybrid for each problem domain.

The parameters of MGA are determined manually as follows: population size (50), tournament size (2) and the number of generations signaling convergence (25), number of genes to mutate (1), maximum chromosome length (5) and maximum number of parents in the mating pool (5).

5.1. Aircraft Landing Problem

The ADHS evolved for ALP is ILS using a simple VNS as an embedded local search (1-level integration). The embedded VNS does not use local search, always accept improving moves and changes NS if no improvement obtained from the current NS. The perturbation of ILS is a random multiple operators procedure similar to the perturbation procedure of Sabar & Kendall (2015). The acceptance criterion accepts improving moves only.

The results obtained by ADHS are compared with those obtained by the state-of-the-art manually designed methods for ALP. These methods include a hybrid particle swarm optimization algorithm with a rolling horizon (RH-HPSO-LS) Girish (2016), a multiple perturbation operators iterated local search with time-varying perturbation (MPO-ILS) Sabar & Kendall (2015), a time discretization algorithm (TDA) with constraints relaxation and a cut algorithm Faye (2015), a hybrid genetic algorithm with tabu search (AGIR) Bencheikh et al. (2013), a hybrid bat algorithm (HBA) Xie et al. (2013), a simulated annealing algorithm (PSA) Awasthi et al. (2013) with an exact procedure for generating an optimal schedule for a given landing sequence and a scatter search algorithm (SS) Pinol & Beasley (2006). To our knowledge, RH-HPSO-LS is the best-performing metaheuristic for ALP (static case).

The results are presented in Table 1 in which the first column presents the instance name, the second column presents the number of runways, the third column presents a unique ID for each instance, the fourth column presents the best-known results (BKR) and from the sixth column to the last column, the best results obtained by the methods are presented in terms of the percentage deviation from BKR which is defined as follows.

$$\Delta = 100 \times \frac{S - \text{BKR}}{\text{BKR}}, \tag{1}$$

where S is the result obtained by the method. Smaller values of Δ indicate better performance. If a method has $\Delta = 0$, the performance of the method is equivalent to the best-known performance. All these methods perform almost identically in the first 25 instances; hence not included in the table.

From the table, the evolved ADHS outperforms all prior methods except RH-HPSO-LS which outperforms ADHS especially in single runway instances. However, both methods tie in 16 instances out of 24. The ADHS performs slightly worse than RH-HPSO-LS in 2 instances. The overall performance of ADHS is competitive with that of the best published method. Please note that the results produced by the MPO-ILS and marked with an asterisk indicate possible errors as pointed out by Girish (2016). This is the case since Girish (2016) was able to solve these instances to optimality using the CPLEX software and the results of the MPO-ILS are lower (better) than the optimal results found by Girish (2016).

Statistical analysis is used in order to ascertain the statistical significance of the results of the ADHS in relation to the results of the manually designed approaches. The application of statistical analysis is done carefully following the guidelines of Derrac et al. (2011). The significance level α is 5%. The method of Xie et al. (2013) is also excluded because it does not provide solutions for the single-runway instances. The four instances for which the MPO-ILS generates erroneous results are excluded from the statistical analysis. When there are three or more methods involved, it is encouraged to use multiple comparison tests with the post-hoc analysis procedures in order to control the family-wise error rate which may be inflated if many individual pairwise comparisons are carried out Derrac et al. (2011). The Friedman multiple comparison test is used. The rankings of the methods computed by the Friedman test is presented

Table 1: ALP: comparison with the State-of-the-art Methods.

Instance	R	ID	BKR	RH-HPSO-LS	ADHS	SS	AGIR	HBA	PSA	TDA	MPO-ILS
Airland9	1	26	5611.7	0	3.7	30.06	22.8	-	1.64	3.22	0
	2	27	444.1	0	0	7.77	1.99	12.47	0	8.48	0.21
	3	28	75.75	0	0	0	0	1.69	0	0	-2.31*
	4	29	0	0	0	0	0	0	0	0	0
Airland10	1	30	12292.2	0	4.0	45.4	15.75	-	9.95	7.93	0.24
	2	31	1143.7	0	0	21.55	15.93	23.06	5.25	2.54	11.14
	3	32	205.21	0	0	17.15	19.75	9.22	0	17.87	-2.53*
	4	33	34.22	0	0	16.74	23.5	0	0	31.91	-6.16*
	5	34	0	0	0	0	0	0	0	0	0
Airland11	1	35	12418.32	0	7.21	17.95	35.2	-	7.92	3.53	-0.05
	2	36	1330.91	0	0.57	26.14	37.4	25.77	5.24	0.38	5.94
	3	37	253.07	0	0	34.92	26.39	10.98	0.03	12.72	7.13
	4	38	54.53	0	0	2.77	0	0	0	3.17	-6.47*
	5	39	0	0	0	0	0	0	0	∞	0
Airland12	1	40	16122.18	0	7.76	22.81	0.74	-	7.59	11.8	0.54
	2	41	1695.62	0	0.10	37.42	16.38	46.39	3.42	5.58	15.67
	3	42	221.97	0	0	53.15	45.95	9.83	5.19	11.93	22.55
	4	43	2.44	0	0	431.55	43.03	0	0	0	39.36
	5	44	0	0	0	0	0	0	0	0	0
Airland13	1	45	37064.11	0	9.56	24.88	20.96	-	16.16	13.04	11.65
	2	46	3920.39	0	1.12	54.56	40.34	32.23	17.18	7.56	39.23
	3	47	673.85	0	0	67.76	69.11	12.06	5.78	16.03	64.5
	4	48	89.95	0	0	157.66	109.52	0.09	0	47.54	3.27
	5	49	0	0	0	∞	∞	0	0	∞	∞

in Table 2 which indicates that HS-HPSO-LS is the best method followed by our ADHS whereas SS is the poorest. The Friedman statistic is 53.98 and the p-value is $7.70E^{-10}$ which strongly suggests the existence of statistically significant differences across the methods. The Friedman test detects differences over multiple comparisons being incapable of establishing proper pairwise comparisons. Therefore, post-hoc analysis procedures are used to conduct pairwise comparisons. The control method is the best-performing one (RH-HPSO-LS). The purpose of the test is to detect which methods (including the evolved ADHS) have a statistically equivalent performance to RH-HPSO-LS. The Holm procedure is used and its results are shown in Table 3. In the table, the first column shows different algorithms (the underlying hypothesis for each algorithm is whether its performance is equivalent to the best method or not), the second column shows the unadjusted p-values, the third column shows the adjusted p-values and the fourth column shows the adjusted significance level. Based on the test results, ADHS and PSA perform statistically equivalent to the best performing method (RH-HPSO-LS). Although not reported here, different post-hoc procedures (Holland, Hochberg, Rom, Finner and Li) lead to the same conclusion drawn from the Holm procedure with respect to the performance of ADHS compared to the top method (RH-HPSO-LS). Based on the statistical analysis, the evolved ADHS has a competitive performance with the state-of-the-art methods for ALP which assures the effectiveness of the proposed approach for automated design.

Table 2: Friedman: average rankings of the algorithms.

Algorithm	RH-HPSO-LS	ADHS	PSA	MPO-ILS	TDA	AGIR	SS
Rankings	1.98	2.80	3.50	3.80	4.33	5.50	6.10

The runtime comparison can only be indicative since it is almost impossible to ensure fair comparison between several methods as it depends on many factors that are difficult to control such as the computer specification, the operating system, the programming language, the performance optimization and the skill level of the programmer. However, we compare ADHS with MPO-ILS in terms of runtime as MPO-ILS is very similar to ADHS in two key aspects.

Table 3: Holm: pairwise comparisons. The control method is RH-HPSO-LS.

Algorithm	unadj. p-value	adj. p-value	adj α
SS	$1.56E^{-9}$	$9.36E^{-9}$	0.0083
AGIR	$2.47E^{-7}$	$1.23E^{-6}$	0.01
TDA	$5.82E^{-4}$	0.0023	0.0125
MPO-ILS	0.008	0.024	0.0167
PSA	0.026	0.052	0.025
ADHS	0.23	0.23	0.05

First, both algorithms are ILS. Second, both algorithms use similar multiple perturbation operators procedure and similar neighborhood structures. The runtime comparison is summarized in Table 4. From the table, ADHS outperforms MPO-ILS substantially in multiple-runway instances whereas it performs worse in single-runway instances.

Table 4: Average runtimes in seconds (rounded to integers).

Instance	26	27	28	29	30	31	32	33	34	35	36	37
MPO-ILS	8	11	11	14	14	16	17	23	34	18	22	34
ADHS	11	6	3	1	24	15	5	3	2	48	28	10
Instance	38	39	40	41	42	43	44	45	46	47	48	49
MPO-ILS	37	55	198	310	402	398	358	486	1011	1123	1181	1152
ADHS	5	3	102	54	18	11	7	699	402	106	45	24

5.2. Two-Dimensional Bin Packing Problem

The ADHS evolved for 2BPP is a sequence consisting of iterated local search (ILS) and variable neighborhood search (VNS). The ILS uses an epsilon-greedy perturbation; see Table 8. The local search of ILS is performed by LS4; see Table 8. The acceptance criterion accepts non-worsening proposals. The VNS is a simple search that generates a candidate solution at random using the current NS (shake), uses LS2, LS3 and LS5 to perform local search and change NS with Change1; see Table 9.

As explained in Section 3, problem-specific design decisions can also be automated by representing each design decision as an additional component of the templates. There are two problem-specific design decisions that are automated which are the *objective function* and *packing heuristics* and their alternative design choices are presented in Table 11.

The performance of ADHS is compared with that of the state-of-the-art methods for 2BBP which are a biased random key genetic algorithm (BRKGA) Gonçalves & Resende (2013), a hybrid evolutionary algorithm (EA-LGFi) Blum & Schmid (2013), a hybrid GRASP/VNS algorithm (GRASP) Parreño et al. (2010), a guided local search algorithm (GLS) Faroe et al. (2003), a tabu search algorithm (TS) Lodi et al. (1999a) and an evolutionary particle swarm optimization algorithm (EPSO) Omar & Ramakrishnan (2013). To our knowledge, BRKGA is the best-performing metaheuristic for 2BPP on the variant of 2BPP considered in this study. Although we are not interested in comparing our method with problem-specific heuristics, we include the set covering heuristic (SCH) Monaci & Toth (2006) in our comparison because it is the best heuristic for the problem.

The results are shown in Table 5 where the performance is measured by the average over the ten instances of each category in each class, see Section 4.2. The best results are highlighted in bold. From the table, the first column shows the classes, the second column shows the instance sizes in each category, the third column shows the lower bounds obtained by Monaci & Toth (2006) and from the fourth column to the last column, the results of the methods are shown. The closer the result of a method to the lower bound, the better the performance of the method is. It can be observed that ADHS finds the best results in 42 categories out of 50 and fails to find the best results in 8 categories. ADHS outperforms ESPO, TS, GLS and SCH, performs almost equivalently to GRASP and performs slightly worse than BRKGA and EA-LFGi. The overall performance of the automated ADHS is competitive with the best-performing manually designed methods.

Table 5: 2BPP: comparison with the state-of-the-art methods.

Class	Size	LB	BRKGA	GRASP	SCH	GLS	TS	EPSO	EA-LGFi	ADHS

Continued on next page

Table 5 – continued from previous page

Class	Size	LB	BRKGA	GRASP	SCH	GLS	TS	EPSO	EA-LGFi	ADHS
Class I	20	7.1	**7.1**	**7.1**	**7.1**	**7.1**	**7.1**	7.29	**7.1**	**7.1**
	40	13.4	**13.4**	**13.4**	**13.4**	**13.4**	13.5	14.51	**13.4**	**13.4**
	60	19.7	**20.0**	**20.0**	**20.0**	20.1	20.1	20.7	**20.0**	**20.0**
	80	27.4	**27.5**	**27.5**	**27.5**	**27.5**	28.2	29.26	**27.5**	**27.5**
	100	31.7	**31.7**	**31.7**	**31.7**	32.1	32.6	32.46	**31.7**	**31.7**
Class II	20	1.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
	40	1.9	**1.9**	**1.9**	**1.9**	**1.9**	2.0	**1.9**	**1.9**	2.0
	60	2.5	**2.5**	**2.5**	**2.5**	**2.5**	2.7	**2.5**	**2.5**	**2.5**
	80	3.1	**3.1**	**3.1**	**3.1**	**3.1**	3.3	**3.1**	**3.1**	**3.1**
	100	3.9	**3.9**	**3.9**	**3.9**	**3.9**	4	**3.9**	**3.9**	**3.9**
Class III	20	5.1	**5.1**	**5.1**	**5.1**	**5.1**	5.5	5.41	**5.1**	**5.1**
	40	9.2	**9.4**	**9.4**	**9.4**	**9.4**	9.7	10.24	**9.4**	**9.4**
	60	13.6	**13.9**	**13.9**	**13.9**	14	14	14.88	**13.9**	**13.9**
	80	18.7	**18.9**	**18.9**	**18.9**	19.1	19.8	20.1	**18.9**	**18.9**
	100	22.1	**22.3**	**22.3**	**22.3**	22.6	23.6	23.87	22.4	**22.3**
Class IV	20	1.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
	40	1.9	**1.9**	**1.9**	**1.9**	**1.9**	**1.9**	**1.9**	**1.9**	**1.9**
	60	2.3	2.5	2.5	2.5	2.5	2.6	2.53	**2.3**	2.4
	80	3.0	**3.1**	**3.1**	3.2	3.3	3.3	3.2	**3.1**	**3.1**
	100	3.7	3.7	3.8	3.8	3.8	4	3.82	**3.7**	**3.7**
Class V	20	6.5	**6.5**	**6.5**	**6.5**	**6.5**	6.6	7.08	**6.5**	**6.5**
	40	11.9	**11.9**	**11.9**	**11.9**	**11.9**	**11.9**	13.04	**11.9**	**11.9**
	60	17.9	**18.0**	**18.0**	**18.0**	18.1	18.2	19.8	**18.0**	**18.0**
	80	24.1	**24.7**	**24.7**	**24.7**	24.9	25.1	26.78	**24.7**	**24.7**
	100	27.9	**28.1**	28.2	28.2	28.8	29.5	29.77	28.4	28.3
Class VI	20	1.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
	40	1.5	**1.6**	1.7	1.7	1.8	1.9	2.1	1.7	1.9
	60	2.1	**2.1**	2.1	2.1	2.2	2.2	2.21	**2.1**	**2.1**
	80	3.0	**3.0**	**3.0**	**3.0**	**3.0**	**3.0**	**3.0**	**3.0**	**3.0**
	100	3.2	3.3	3.4	3.4	3.4	3.4	3.41	**3.2**	**3.2**
Class VII	20	5.5	**5.5**	**5.5**	**5.5**	**5.5**	**5.5**	6.11	**5.5**	**5.5**
	40	10.9	**11.1**	**11.1**	**11.1**	11.3	11.4	11.8	**11.1**	**11.1**
	60	15.6	**15.8**	15.9	**15.8**	15.9	16.2	16.99	15.9	**15.8**
	80	22.4	23.2	**23.2**	**23.2**	**23.2**	**23.2**	24.44	**23.2**	**23.2**
	100	26.9	**27.1**	**27.1**	**27.1**	27.5	27.7	29.13	**27.1**	**27.1**
ClassVIII	20	5.8	**5.8**	**5.8**	**5.8**	**5.8**	**5.8**	6.38	**5.8**	**5.8**
	40	11.2	**11.3**	**11.3**	**11.3**	11.4	11.4	12.24	**11.3**	**11.3**
	60	15.9	**16.1**	**16.1**	16.2	16.3	16.2	17.47	**16.1**	16.2
	80	22.3	**22.4**	**22.4**	**22.4**	22.5	22.6	24.33	**22.4**	**22.4**
	100	27.4	27.8	27.8	27.9	28.1	28.4	29.81	**27.7**	27.8
	20	14.3	**14.3**	**14.3**	**14.3**	**14.3**	**14.3**	**14.3**	**14.3**	**14.3**
	40	27.8	**27.8**	**27.8**	**27.8**	**27.8**	**27.8**	28.41	**27.8**	**27.8**

Continued on next page

Table 5 – continued from previous page

Class	Size	LB	BRKGA	GRASP	SCH	GLS	TS	EPSO	EA-LGFi	ADHS
Class IX	60	43.7	**43.7**	**43.7**	**43.7**	**43.7**	43.8	44.31	**43.7**	**43.7**
	80	57.7	**57.7**	**57.7**	**57.7**	**57.7**	**57.7**	59.2	**57.7**	**57.7**
	100	69.5	**69.5**	**69.5**	**69.5**	**69.5**	**69.5**	70.82	**69.5**	**69.5**
Class X	20	4.2	**4.2**	**4.2**	**4.2**	**4.2**	4.3	4.73	**4.2**	**4.2**
	40	7.4	**7.4**	**7.4**	**7.4**	**7.4**	7.5	7.96	**7.4**	**7.4**
	60	9.8	**10.0**	**10.0**	10.1	10.2	10.4	10.46	10.1	10.1
	80	12.3	**12.8**	12.9	**12.8**	13.0	13.0	13.11	**12.8**	**12.8**
	100	15.3	**15.8**	15.9	15.9	16.2	16.6	16.31	16.0	15.9
Sum		7173	7234	7241	7243	7284	7360	7600.7	7239	7241

As done in Section 5.1, statistical analysis is used to ascertain whether the evolved ADHS is comparable with the manually designed approaches or significantly worse. Again, the guidelines of Derrac et al. (2011) are followed as precise as possible. The significance level α is 5%. The Friedman test is used to detect statistical differences across the methods (multiple comparisons). The rankings of the methods computed by the Friedman is presented in Table 6 which reveal that our method is the third best method. The rankings also reveal that performance gap across the whole dataset between the top five methods (including our ADHS) is rather small. The Friedman statistic is 128.29 and the p-value is $6.23E^{-10}$ which strongly suggests the existence of statistically significant differences among the methods. Again, post-hoc analysis procedures are used to detect pairwise significant differences. The control method is the best-performing one (BRKGA). The purpose of the test is to detect which methods (including the evolved ADHS) have a statistically equivalent performance to BRKGA. The Holm procedure is used and its results are shown in Table 7. Based on the test results, EA-LGFi, GRASP, SCH and the evolved ADHS perform statistically equivalent to the best performing method BRKGA. Although not reported here, different post-hoc procedures (Holland, Hochberg, Rom, Finner and Li) lead to the same conclusion drawn from the Holm procedure. Based on the statistical analysis, the evolved ADHS performs no worse than the state-of-the-art methods for 2BPP which assures the competitiveness of the proposed

Table 6: Friedman: average rankings of the methods.

Algorithm	BRKGA	EA-LGFi	ADHS	GRASP	SCH	GLS	TS	EPSO
Rankings	3.24	3.47	3.60	3.61	3.74	4.95	6.28	7.12

Table 7: Holm: pairwise comparisons. The control method is BRKGA.

Algorithm	unadj. p-value	adj. p-value	adj α
EPSO	$1.21E^{-15}$	$8.45E^{-15}$	0.007
TS	$3.26E^{-10}$	$1.96E^{-9}$	0.008
GLS	$4.05E^{-4}$	0.002	0.01
SCH	0.30	1.21	0.0125
GRASP	0.44	1.33	0.0167
ADHS	0.45	1.33	0.025
EA-LGFi	0.63	1.33	0.05

approach for automated design compared to the traditional manual approach.

Further, from the result of the statistical analysis, it can be observed that relatively recently proposed metaheuristics (BRKGA, EA-LGFi, GRASP and our ADHS) perform well and equivalently whereas early proposed metaheuristics (TS and GLS) perform poorly and significantly worse. This may suggest that metaheuristics have advanced well enough *in this variant* of 2BPP and/or this standard benchmark dataset is not hard enough to distinguish between methods which may raise the need for a new benchmark set.

The average runtime of ADHS on 2BPP ranges from 0 seconds on the smallest instance to less than 20 seconds on the largest.

It can be noted that for both problem domains the memetic algorithm was not included in the best evolved hybrid, however for other problem domains it may be included and this will be examined as part of future work. We conclude this section by making a remark. In both domains (ALP and 2BPP), it is found that ADHS perform statistically equivalent to the best-performing metaheuristics on standard benchmark datasets. A lacking feature of the manual design is reusability, i.e. the manual design has to be repeated whenever a new problem domain is encountered. On the contrary, the automated approach is reusable, i.e. the same approach is used to automatically design solvers for ALP and 2BPP at a cost of minimal adjustments, such as problem-specific implementation and supplying new training instances.

6. Conclusion

This paper investigates the feasibility of the automated design of relay hybrid metaheuristics using a meta-genetic algorithm working on the space of configurations of hybrid metaheuristics. The proposed approach extends prior studies Hassan & Pillay (2017, 2018) which demonstrate the effectiveness of meta-genetic algorithms for the design of hybrid metaheuristics. In Hassan & Pillay (2017, 2018), a meta-genetic algorithm is used to automatically determine which sequence of metaheuristics to use, with what parameter values (parameter tuning) and in what order the metaheuristics should be applied. This paper extends these prior studies by automating the design of each metaheuristic in the sequence in addition to automating the sequential hybridization and parameter tuning. This is achieved by defining algorithmic templates consisting of components representing key design decisions such as what acceptance criterion to use. A meta-genetic algorithm is used to determine the suitable components for each metaheuristic in the hybrid, the best sequential way of combining the metaheuristics and the parameter values for each metaheuristic in the sequence. The proposed automated approach is evaluated by using it to automate the design of hybrid metaheuristics for two hard, well-known problems: the aircraft landing problem and the two-dimensional bin packing problem. The automatically designed hybrid metaheuristics were found to perform competitively and in some cases better than the previously proposed, best-performing hybrid metaheuristics which were designed manually. This study has illustrated the potential of automating the design of relay hybrid metaheuristics and the individual metaheuristics comprising the hybrid using two different problem domains. Given the success for these two very different problem domains, future work will apply the approach to additional problems such as logistics problems.

In the future, this work will be extended by considering the automation of hybrid metaheuristics without relying on predefined templates (structures). In this case, the automated approach is responsible for discovering the optimal or near optimal structure as part of the design process. There has been prior research in unifying the view of metaheuristics; see for instance Raidl (2006). Once a unified view is established, a grammar can be used to describe metaheuristics generally based on the unified view without a particular reference to a specific template. As a result, a grammatical evolution algorithm, a genetic programming algorithm or an algorithm configuration tool can be used to navigate the search space of hybrid metaheuristics as defined by the grammar. The advantage of this bottom-up approach to the design of hybrid metaheuristics is its ability to come up with new metaheuristics that might have not been discovered before whereas the downside is its complexity and the possible limitation as a result of the unified view, the grammar and/or both. In this study, an equal amount of computational budget, i.e. iterations, have been allocated to each metaheuristic in the hybrid. Future work will also investigate automating the design of the time slices allocated to each metaheuristic in the hybrid.

Future work will also examine the reusability of the hybrid metaheuristics produced. The idea would be to automate the design of the hybrid metaheuristics on a training set and apply to a test set to see how well the hybrid metaheuristic performs. The study will examine whether the hybrid metaheuristics are reusable for different classes of problems or across problem classes. This work will also investigate the effect of evolving a hybrid for each problem instance.

There are various alternatives to this approach such as portfolio algorithms Calderín et al. (2017) and agent-based cooperative approaches Moreno et al. (2016). Future work will also investigate a comparison of such alternatives with the approach presented in the paper.

Acknowledgment

The authors would like to thank the reviewers for the invaluable comments to improve the quality of the paper. The authors would like to thank the National Research Foundation (NRF), South Africa for funding this research project [Grant number: 99811]. The authors would like to thank the Center for High Performance Computing (CHPC), South Africa for providing access to the cluster.

References

Awasthi, A., Kramer, O., & Lässig, J. (2013). Aircraft landing problem: An efficient algorithm for a given landing sequence. In *IEEE 16th International Conference on Computational Science and Engineering* (pp. 20–27). IEEE.

Battiti, R., Brunato, M., & Mascia, F. (2008). *Reactive search and intelligent optimization*. Springer Science & Business Media.

Beasley, J., Krishnamoorthy, M., Sharaiha, Y., & Abramson, D. (2004). Displacement problem and dynamically scheduling aircraft landings. *Journal of the operational research society*, *55*, 54–64.

Beasley, J. E., Krishnamoorthy, M., Sharaiha, Y. M., & Abramson, D. (2000). Scheduling aircraft landings—the static case. *Transportation science*, *34*, 180–197.

Bencheikh, G., El Khoukhi, F., Baccouche, M., Boudebous, D., Belkadi, A., & Ouahman, A. A. (2013). Hybrid algorithms for the multiple runway aircraft landing problem. *International Journal on Computer Science & Applications*, *10*, 53–71.

Berkey, J. O., & Wang, P. Y. (1987). Two-dimensional finite bin-packing algorithms. *Journal of the operational research society*, *38*, 423–429.

Bhanu, S. M. S., & Gopalan, N. (2008). A hyper-heuristic approach for efficient resource scheduling in grid. *International Journal of Computers Communications & Control*, *3*, 249–258.

Blum, C., & Schmid, V. (2013). Solving the 2D bin packing problem by means of a hybrid evolutionary algorithm. *Procedia Computer Science*, *18*, 899–908.

Burke, E. K., 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*, *64*, 1695–1724.

Calderín, J. F., Masegosa, A. D., & Pelta, D. A. (2017). An algorithm portfolio for the dynamic maximal covering location problem. *Memetic Computing*, *9*, 141–151.

Davis, L. (1991). *Handbook of genetic algorithms*. Van Nostrand Reinhold .

Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, *1*, 3–18.

Dioşan, L., & Oltean, M. (2009). Evolutionary design of evolutionary algorithms. *Genetic Programming and Evolvable Machines*, *10*, 263–306.

Drake, J. H., Kililis, N., & Özcan, E. (2013). Generation of VNS components with grammatical evolution for vehicle routing. In *European Conference on Genetic Programming* (pp. 25–36). Springer.

Eiben, A. E., Raue, P.-E., & Ruttkay, Z. (1994). Genetic algorithms with multi-parent recombination. In *International Conference on Parallel Problem Solving from Nature* (pp. 78–87). Springer.

Ernst, A. T., Krishnamoorthy, M., & Storer, R. H. (1999). Heuristic and exact algorithms for scheduling aircraft landings. *Networks*, *34*, 229–241.

Faroe, O., Pisinger, D., & Zachariasen, M. (2003). Guided local search for the three-dimensional bin-packing problem. *Informs journal on computing*, *15*, 267–283.

Faye, A. (2015). Solving the aircraft landing problem with time discretization approach. *European Journal of Operational Research*, *242*, 1028–1038.

Garcia-Villoria, A., Salhi, S., Corominas, A., & Pastor, R. (2011). Hyper-heuristic approaches for the response time variability problem. *European Journal of Operational Research*, *211*, 160–169.

Gendreau, M., & Potvin, J. (2010). *Handbook of Metaheuristics*. Springer US.

Girish, B. (2016). An efficient hybrid particle swarm optimization algorithm in a rolling horizon framework for the aircraft landing problem. *Applied Soft Computing*, *44*, 200–221.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company.

Gomes, C. P., & Selman, B. (2001). Algorithm portfolios. *Artificial Intelligence*, *126*, 43–62.

Gonçalves, J. F., & Resende, M. G. (2013). A biased random key genetic algorithm for 2D and 3D bin packing problems. *International Journal of Production Economics*, *145*, 500–510.

González, B., Valdez, F., Melin, P., & Prado-Arechiga, G. (2015). Fuzzy logic in the gravitational search algorithm enhanced using fuzzy logic with dynamic alpha parameter value adaptation for the optimization of modular neural networks in echocardiogram recognition. *Applied Soft Computing*, *37*, 245–254.

Grobler, J., Engelbrecht, A. P., Kendall, G., & Yadavalli, V. (2010). Alternative hyper-heuristic strategies for multi-method global optimization. In *IEEE Congress on Evolutionary Computation* (pp. 1–8). IEEE.

Grobler, J., Engelbrecht, A. P., Kendall, G., & Yadavalli, V. (2012). Investigating the use of local search for improving meta-hyper-heuristic performance. In *IEEE Congress on Evolutionary Computation* (pp. 1–8). IEEE.

Hansen, P., Mladenović, N., & Pérez, J. A. M. (2010). Variable neighbourhood search: methods and applications. *Annals of Operations Research*, *175*, 367–407.

Hassan, A., & Pillay, N. (2017). A meta-genetic algorithm for hybridizing metaheuristics. In *Portuguese Conference on Artificial Intelligence* (pp. 369–381). Springer.

Hassan, A., & Pillay, N. (2018). An improved meta-genetic algorithm for hybridizing metaheuristics. In *IEEE Congress on Evolutionary Computation* (pp. 1453–1460). IEEE.

Hopper, E. (2000). *Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods*. Ph.D. thesis University of Wales. Cardiff.

Hutter, F., Hoos, H. H., & Stützle, T. (2007). Automatic algorithm configuration based on local search. In *The AAAI Conference on Artificial Intelligence* (pp. 1152–1157). AAAI.

Keller, R. E., & Poli, R. (2007). Linear genetic programming of parsimonious metaheuristics. In *IEEE Congress on Evolutionary Computation* (pp. 4508–4515). IEEE.

Lai, K., & Chan, J. W. (1997). Developing a simulated annealing algorithm for the cutting stock problem. *Computers & industrial engineering*, *32*, 115–127.

Lodi, A., Martello, S., & Monaci, M. (2002a). Two-dimensional packing problems: A survey. *European journal of operational research*, *141*, 241–252.

Lodi, A., Martello, S., & Vigo, D. (1999a). Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, *112*, 158–166.

Lodi, A., Martello, S., & Vigo, D. (1999b). Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, *11*, 345–357.

Lodi, A., Martello, S., & Vigo, D. (2002b). Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, *123*, 379–396.

López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., & Birattari, M. (2011). *The irace package, iterated race for automatic algorithm configuration*. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium.

López-Ibánez, M., Kessaci, M.-E., & Stützle, T. (2017). *Automatic design of hybrid metaheuristics from algorithmic components*. Technical Report TR/IRIDIA/2017-012, IRIDIA, Université Libre de Bruxelles, Belgium.

Lopez-Ibanez, M., & Stutzle, T. (2012). The automatic design of multiobjective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, *16*, 861–875.

Marmion, M.-E., Mascia, F., López-Ibánez, M., & Stützle, T. (2013). Automatic design of hybrid stochastic local search algorithms. In *International Workshop on Hybrid Metaheuristics* (pp. 144–158). Springer.

Martello, S., & Vigo, D. (1998). Exact solution of the two-dimensional finite bin packing problem. *Management science*, *44*, 388–399.

Monaci, M., & Toth, P. (2006). A set-covering-based heuristic approach for bin-packing problems. *INFORMS Journal on Computing*, *18*, 71–85.

Moreno, M., Rosete, A., & Pavón, J. (2016). An agent based approach for the implementation of cooperative proactive s-metaheuristics. *Expert Systems with Applications*, *63*, 344–374.

Moscato, P. et al. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, *826*, 1989.

Mühlenbein, H., Gorges-Schleuter, M., & Krämer, O. (1988). Evolution algorithms in combinatorial optimization. *Parallel Computing*, *7*, 65–85.

Nyathi, T., & Pillay, N. (2018). Comparison of a genetic algorithm to grammatical evolution for automated design of genetic programming classification algorithms. *Expert Systems with Applications*, *104*, 213–234.

Olivas, F., Valdez, F., Melin, P., Sombra, A., & Castillo, O. (2019). Interval type-2 fuzzy logic for dynamic parameter adaptation in a modified gravitational search algorithm. *Information Sciences*, *476*, 159–175.

Oltean, M. (2005). Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation*, *13*, 387–410.

Omar, M. K., & Ramakrishnan, K. (2013). Solving non-oriented two dimensional bin packing problem using evolutionary particle swarm optimisation. *International Journal of Production Research*, *51*, 6002–6016.

Parreño, F., Alvarez-Valdés, R., Oliveira, J., & Tamarit, J. M. (2010). A hybrid grasp/vnd algorithm for two-and three-dimensional bin packing. *Annals of Operations Research*, *179*, 203–220.

Pinol, H., & Beasley, J. E. (2006). Scatter search and bionomic algorithms for the aircraft landing problem. *European Journal of Operational Research*, *171*, 439–462.

Raidl, G. R. (2006). A unified view on hybrid metaheuristics. In *International Workshop on Hybrid Metaheuristics* (pp. 1–12). Springer.

Rice, J. R. (1976). The algorithm selection problem. *Advances in computers*, *15*, 65–118.

Sabar, N. R., Ayob, M., Kendall, G., & Qu, R. (2013). Grammatical evolution hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, *17*, 840–861.

Sabar, N. R., & Kendall, G. (2015). An iterated local search with multiple perturbation operators and time varying perturbation strength for the aircraft landing problem. *Omega*, *56*, 88–98.

Salehipour, A., Modarres, M., & Naeni, L. M. (2013). An efficient hybrid meta-heuristic for aircraft landing problem. *Computers & Operations Research*, *40*, 207–213.

Sevaux, M., Sörensen, K., & Pillay, N. (2018). Adaptive and multilevel meta-heuristics. *Handbook of Heuristics*, (pp. 1–19).

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*. John Wiley & Sons.

Tavares, J., & Pereira, F. (2012). Automatic design of ant algorithms with grammatical evolution. In *European Conference on Genetic Programming* (pp. 206–217). Springer.

Tavares, J., & Pereira, F. B. (2011). Towards the development of self-ant systems. In *The 13th Annual Conference on Genetic and Evolutionary Computation* (pp. 1947–1954). ACM.

Tsai, C.-W., Song, H.-J., & Chiang, M.-C. (2012). A hyper-heuristic clustering algorithm. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 2839–2844). IEEE.

Valdez, F., Melin, P., & Castillo, O. (2014). A survey on nature-inspired optimization algorithms with fuzzy logic for dynamic parameter adaptation. *Expert systems with applications*, *41*, 6459–6466.

Xie, J., Zhou, Y., & Zheng, H. (2013). A hybrid metaheuristic for multiple runways aircraft landing problem based on bat algorithm. *Journal of Applied Mathematics*, *2013*.

Appendix A

This appendix presents the problem-specific design choices and problem-independent design choices for the iterated local search, variable neighborhood search and the memetic algorithm.

Design Choices for Iterated Local Search

The ILS template consists of three basic components which are the perturbation, local search and acceptance criterion. The design choices for each component are presented in Table 8.

Table 8: Design choices for the three components of the iterated local search.

Problem	Choice	Description
Design Choices for Peturbation		
ALP	Move1	Pick a plane at random and move it to a different position in the same runway.
	Move2	Pick a plane at random and change its runway.
	Swap1	Select two planes at random from the same runway and swap their positions in the landing sequence.
	Swap2	Same as Swap1 but the planes are chosen from different runways.
2BPP	Repack1	Sort bins in a non-increasing order of their occupancy. Remove the last k packed items and add them to a packing queue which contains the items that are not packed yet. Shuffle and repack items in the queue.
	Repack2	Sort bins as in Repack1. Remove the last k packed items from each bin and add them to the packing queue. Shuffle and repack items in the queue.
	SplitHoriz	Sort bins as in Repack1. Choose the N_s bins where N_s is chosen at random from the range $[N/2, N]$ and N is the total number of used bins. Choose at random which part of the bins to empty (upper/lower). Choose a horizontal split axis. Removes item in the chosen part (upper/lower) and add them to the packing queue. Shuffle and repack.
	SplitVert	Same as in SplitHoriz with the exception that the split axis is vertical.
Problem-Independent	Random	Choose a perturbtive operator at random and apply it to the current solution.
	RouletteWheel	Choose a perturbative operator probabilistically where the probability of choosing a perturbative operator is proportional to its value/merit.
	Greedy	Choose a perturbative operator that has the best value.
	EpsilonGreedy	Choose the best perturbative operator with probability $1-\epsilon$; otherwise, choose a perturbative operator at random.
	Cyclic	Choose perturbative operators on a rotational basis where after every q iterations the same perturbative operator is chosen.

Continued on next page

Table 8 – continued from previous page

Problem	Choice	Description
		Design Choices for Local Search
ALP	LS1	Choose a runway at random. Iterate over all planes and change their position within the same runway. Accept first improving moves.
	LS2	Same as LS1 but accept best improving moves.
	LS3	Choose two runways at random. Iterate over all planes of the first runway and schedule them on the other runway. Accept first-improving moves.
	LS4	Same as LS3 but accept best-improving moves.
	LS5	Choose a runway at random. Try all possible swaps within the same runway. Accept first-improving moves.
	LS6	Same as LS5 but accept best-improving moves.
	LS7	Choose two runways at random. Try all possible swaps between the two runways. Accept first-improving moves.
	LS8	Same as LS7 but accept best-improving moves.
	LS9	Optimal block procedure described in Girish (2016).
2BPP	LS1	Sort bins as in Repack1. Remove all items from every two consecutive bins and add them to the packing queue. Repack the items from the queue in the current two bins with the condition that the first item to be packed is chosen from the items that were already in the queue before emptying the two bins.
	LS2	Same as LS1 but consider every three consecutive bins.
	LS3	Same as LS1 but consider every four consecutive bins.
	LS4	Same as LS1 but consider every two possible bins.
Problem-independent	VND	Variable neighborhood descent as described in Hansen et al. (2010).
	RVND	Same as VND but the order by which the neighborhood structures are visited is randomly changed every time RVND is called.
	SA	Simulated annealing as described in Gendreau & Potvin (2010).
	RCR	A template can call itself with different design choices causing an inner instance of the same template to be used as an embedded local search within another instance.
		Acceptance Criterion
Problem-Independent	AcceptAll	Accept all proposals.
	AcceptImproving	Accept improving proposals only.
	AcceptNonWorse	Accept non-worsening proposals.
	ThresholdAccept	Accept improving proposals always. Worsening proposals are accepted if they are at most $\delta\%$ worse than the current solution.
	MetropolisAccept	Accept improving proposal always. Worsening proposals are accepted with a probability following Boltzmann distribution.
	LateAccept	Accepts candidate solutions that are no worse than that solution which "was" the current solution M iterations ago.

Design Choices for Variable Neighborhood Search

The basic components of the variable neighborhood search (VNS) are shake, local search and neighborhood change. The shake component has only one design choice which generate a new solution using the current neighborhood structure. The design choices for the local search component of VNS is the same as the design choices for the local search component of ILS which are presented in Table 8. The design choices for the neighborhood change component are presented in Table 9.

Table 9: Variable neighborhood search: design choices for the neighborhood change.

Problem	Choice	Description
Problem-independent	Change1	If current neighborhood k leads to an improving proposal, accepts it and returns to the first neighborhood structure $k = 1$; otherwise, move the next neighborhood $k + 1$.
	Change2	Same as Change1 but accept worsening proposals with probability p.
	Change3	Same as Change1 but accept worsening proposals that are no worse than δ% of the current solution.
	Change4	Same as Change1 but move to a random neighborhood if there is no improvement instead of moving to the next neighborhood $k + 1$.

Design Choices for Memetic Algorithm

The basic components of the memetic algorithms (MA) templates are selection, crossover, mutation and local search. The design choices for the local search component are the same as the ones presented in Table 8 with the exception of recursion. Recall that for recursion, a template calls itself causing an inner instance of the template (metaheuristic) to be used as embedded local search. As mentioned in Section 3, for practical reasons, a population-based method is not allowed to be used as embedded local search. Therefore, all local searches mentioned in Table 8 are available for MA except recursion. The design choices for the components of MA are presented in Table 10.

Table 10: Design choices for the components of the memetic algorithms.

Problem	Choice	Description
Design Choices for Selection		
Prob.-independ	Tournament	N_T individuals are chosen at random from the population. The best individual is selected for mating.
	RouletteWheel	The probability of selecting an individual is proportional to its fitness.
Design Choice for Crossover		
Prob.-Independ.	PMX	Partially Mapped Crossover. Please refer to Goldberg (1989) for explanation.

continued on next page

Table10– continued from previous page

Problem	Choice	Description
	MPX	Maximum Preservative Crossover. Please refer to Mühlenbein et al. (1988) for explanation.
	OCGS	Order crossover with Gene Swapping. Please refer to Davis (1991) for explanation.
Design Choices for Mutation		
Problem-independ.	MFNG1	Mutate Fixed Number of Genes: select μ genes at random and swap their positions in the chromosomes.
	MFNG2	Same as MFNG1 but differs in restricting the choice of the two genes to be swapped to be within a window of length l.
	CPG	Change Preserved Genes: consider the preserved genes in both parents and mutate each with probability P_μ.

Problem-Specific Design Choices

As mentioned in Section 3, hard-to-make design decisions can also be automated. For the ALP, there is no such design decisions to automate. For the 2BPP, there are two major design decisions which are *packing heuristics* and *objective function*. These design decision are made when initializing a metaheuristic from one of the templates. For each design decision, there are design choices which are presented in Table 11.

Table 11: Problem-specific design choices for 2BPP.

Decision	Choice	Description
Design Choices for Objective Function		
Objective function	Occupancy	This objective function is defined as $f_A = N + A$ where N is the number of used bins and A is the ratio of the area of the items in the least filled bin to the total area of the bin. Note that because $0 \leq A \leq 1$, f_A favors solution with lower number of bins. Ties are broken using A in case of two solutions with the same N in favor for the solution with the lower value of A. This objective function is used by Gonçalves & Resende (2013)

continued on next page

Table11– continued from previous page

Decision	Choice	Description
	Structure	This objective function is defined as $f_T = N + (1 - T)$ where N is the number of used bins and T is the average normalized touching perimeter across all bins. The normalized touching perimeter of a bin is the ratio between the sum of the perimeters shared amongst the items and/or the bin's sides and the sum of the total perimeters of all items in the bin. Note that because $0 \leq T \leq 1$, f_T favors solution with lower number of bins and breaks ties using T in case of two solutions with the same N in favor for the solution with the higher value of T, i.e. solution with good packing structure. *This objective function has never been used before to our knowledge.*
Design Choices for Packing Heuristics		
Packing heuristics	BestAreaFit	Best Area Fit: pack the current item in the maximal space that results in the least wasted area. This packing heuristic favors packing that results in the least wasted areas.
	TouchingPerimeter	Touching Perimeter: pack the current item in the maximal space which maximizes the shared perimeters between the current item and the already-packed items and/or the bin sides. This packing heuristic favors packing that does not trap small areas between items.
	TopRightCorner	Pack the current item in the maximal space that maximizes the distance between the top right corner of the item and that of the bin. This packing heuristic favors compact packing.

Please note that all the maximal space data structure Lai & Chan (1997) is used to track empty areas in the bins.

Appendix B

In this appendix we provided initial solution construction methods used for the ALP and 2BPP.

Initial Solution for ALP

The initial solution for the single-point searches is created as follows. First, a random landing sequence is created using Algorithm 4. Then, the optimal block procedure Girish (2016). is used to generate a complete landing schedule. For multiple runway instances, the plane is assigned to the runway that minimizes the cost. Please note that landing sequences generated by Algorithm 4 can lead to infeasible solution in which case the algorithm is tried three times and upon the failure for the third times, a deterministic procedure is used as in Salehipour et al. (2013). For the MA, the initial population is comprised of the landing sequences generated by Algorithm 4.

Please note that there is a definite need for a procedure for generating random landing sequences for the ALP as our experiments confirm that a pure random generation procedure leads to infeasible solutions with very high probability. Thus, we developed a novel procedure for random landing sequence generation which is described by Algorithm 4 in which an aircraft at position i in S_T is denoted by A_i. The experiments show that RLSG generates feasible landing sequence with high probability if $L = 5$ for instances with $n < 500$ and $L = 3$ for instances with sizes $n \geq 500$. If 3 trials are used, the probability of generating a feasible solution in any one of the trials is very close to 1. The probability of generating the same sequence (generating duplicates) is very close to 0.

Algorithm 4 A procedure for generating random landing sequences.

Require: Maximum window length L

function RLSG(L)
  $LS \leftarrow$ an initial landing sequence (an array filled with minus ones).
  $S_T \leftarrow$ sort aircraft according to their target landing time.
  $i \leftarrow 0$
  repeat
    $W_i \leftarrow$ aircraft in S_T from position i to position $i + L$ (or to position $i + n$ if $i + L > n$).
    $L'_i \leftarrow$ the number of aircraft in W_i that can be scheduled earlier than A_i.
    $W'_i \leftarrow$ aircraft from position i to position $i + L'_i$ in S_T.
    if $L'_i = 0$ **then**
      $LS[i] \leftarrow A_i$
    else
      for each A_k in W'_i **do**
        $J_k \leftarrow$ set of positions within the range $[i, i + L'_i]$ that can accommodate aircraft A_k without conflicting with any aircraft in the window W'_i.
        $j \leftarrow$ pick an element from J_k at random
        $LS[j] = A_k$
      end for
    end if
    $i \leftarrow i + L'_i + 1$
  until $i > n$
end function

Initial Solution for 2BPP

The initial solution for single-point searches is created using BestAreaFit heuristic presented in Table 11. For the MA, the initial population is represented by a sequence of integers determining the order by which the items are supposed to be packed where each item is identified by a unique integer.