GRAHF: a hyper-heuristic framework for evolving heterogeneous island model topologies

Jonathan Wurth, Helena Stegherr, Michael Heider, Jörg Hähner

Angaben zur Veröffentlichung / Publication details:

Wurth, Jonathan, Helena Stegherr, Michael Heider, and Jörg Hähner. 2024. "GRAHF: a hyper-heuristic framework for evolving heterogeneous island model topologies." In *GECCO '24: proceedings of the Genetic and Evolutionary Computation Conference, Melbourne, VIC, Australia, July 14-18, 2024*, edited by Xiaodong Li and Julia Handl, 1054–63. New York, NY: ACM. https://doi.org/10.1145/3638529.3654136.

GRAHF: A Hyper-Heuristic Framework for Evolving Heterogeneous Island Model Topologies

Jonathan Wurth
jonathan.wurth@uni-a.de
Universität Augsburg
Augsburg, Germany

Helena Stegherr
helena.stegherr@uni-a.de
Universität Augsburg
Augsburg, Germany

Michael Heider
michael.heider@uni-a.de
Universität Augsburg
Augsburg, Germany

Jörg Hähner
joerg.haehner@uni-a.de
Universität Augsburg
Augsburg, Germany

ABSTRACT

Practitioners frequently encounter the challenge of selecting the best optimization algorithm from a pool of options. However, why not, rather than selecting a single algorithm, let evolution determine the optimal combination of all algorithms? In this paper, we present an approach to algorithm design inspired by a well-known traditional method for coarse-grained hybridization: the heterogeneous island model. Our hyper-heuristic framework represents island models as graphs and identifies optimal island topologies and parameters for specific sets of problem instances. Since the framework operates at the level of metaheuristic algorithms rather than components and incorporates a configuration mechanism directly into the search, it combines concepts from algorithm design, selection, and configuration. The proposed framework is investigated on 24 training sets of varying difficulty and demonstrates its ability to discover complex hybrids. A post-evaluation on real-world constrained optimization problems shows a significant improvement over the algorithms on their own. These results suggest that it is a promising way to design hybrid metaheuristics with minimal manual intervention, given representative training instances, a set of optimization algorithms, and sufficient computational resources.

CCS CONCEPTS

• **Theory of computation → Algorithm design techniques**; **Genetic programming**; *Bio-inspired optimization.*

KEYWORDS

automated algorithm design, hyper-heuristics, island model

ACM Reference Format:
Jonathan Wurth, Helena Stegherr, Michael Heider, and Jörg Hähner. 2024. GRAHF: A Hyper-Heuristic Framework for Evolving Heterogeneous Island Model Topologies. In *Genetic and Evolutionary Computation Conference (GECCO '24), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3638529.3654136

GECCO '24, July 14–18, 2024, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0494-9/24/07.
https://doi.org/10.1145/3638529.3654136

1 INTRODUCTION

A fundamental strength of metaheuristics is their versatility as general search methodologies, allowing customization of components to address different optimization problems. Over the past few decades, various metaheuristics and problem-specific adaptations have emerged across diverse communities, intersecting fields such as evolutionary computation, engineering design, computational biology, machine learning, and robotics [54]. Although they provide no guarantees regarding solution optimality, metaheuristics have proven highly effective in tackling real-world problems, outperforming many exact methods [12].

Tailoring algorithms to specific problems is an increasingly important task, and is supported by insights such as the No-Free-Lunch Theorem [58]. Usually, when identifying the most appropriate algorithm for a finite subset of problems, the assumptions of the theorem are not satisfied, leading to the possibility of a "free lunch" [41]. Instances in real-world problems also often adhere to specific distributions and exhibit recurring characteristics (e.g., hardware model boolean satisfiability problem instances [17]). The search for superior algorithms within specific problem subsets is therefore well motivated.

Developing such effective algorithms for specific optimization problems typically involves a process of trial and error, comparing different algorithmic components and parameter values in a search guided by the designer's expertise and intuition. Stützle and López-Ibáñez emphasize in [51] that this manual approach, besides being labor-intensive and complex, has inherent flaws: (1) Only a relatively small number of design alternatives can be explored, (2) the development process often lacks transparency, hiding both successful and unsuccessful design choices, and (3) designers drawing from experience may tend to over-generalize, e.g., by assuming independence between certain algorithm components and parameters. From this and the fact that finding a good metaheuristic algorithm on a distribution of instances can itself be treated as an optimization problem, there is substantial motivation for applying automated methods to ensure a controlled and reproducible search. The increasing popularity of automated algorithm design in recent years is also accelerated by the increase in available computing power, given the resource-intensive nature of evaluating algorithm performance [61].

In this paper, we propose GRAHF[1], a novel method for the automated design of heterogeneous island model topologies, i.e., coarse-grained hybrid metaheuristics. It relies on the island model as a general hybridization framework, a model initially developed for distributing the execution of an evolutionary algorithm across a cluster of nodes [6]. We introduce a graph-based genetic programming (GP) hyper-heuristic that is able to find island topologies that excel at solving specific distributions or subsets of problem instances. By splitting our design space into a higher-level algorithm design task and a lower-level parameter configuration task, we can apply arbitrary configuration mechanisms to identify optimal parameters along with a suitable overall structure. We additionally present a robust performance metric for estimating the performance of an algorithm on a training set of problem instances.

The rest of the paper is structured as follows: the next section briefly highlights key areas relevant to GRAHF, including the island model, graph-based evolutionary algorithms, and automated algorithm design (AAD), after which Section 3 gives an overview of related work. In Section 4, we present the framework in-depth, detailing the design space, the hyper-heuristic, and the measure for estimating the performance of the generated island algorithms. GRAHF is applied to various training sets of problem instances, the experimental setup and results of which are discussed in Section 5. A post-evaluation of elitist topologies generated by these experiments and non-hybrid reference algorithms is conducted in Section 6. The paper concludes with Section 7, where an outlook on future research is given and the findings of this paper are summarized.

2 BACKGROUND

The island model is a technique for coarse-grained parallelization of metaheuristics, originally called the Distributed Genetic Algorithm [6]. The main idea is to divide the population of an algorithm into several subpopulations or "islands" and evolve each island on a separate processor; an alternative view is that several metaheuristics with a smaller population are executed in parallel. While the islands evolve primarily in isolation (similar to parallel algorithm portfolios), individuals can occasionally migrate between islands, allowing the exchange of information. This allows each island to explore a different region of the search space, greatly increasing the potential for diversity [35]. Diversity, and often also performance, can be further improved by using different parameter configurations or even different algorithms on each island, which is then referred to as a *heterogeneous* island model. The main property of an island model is its topology, which describes the islands and the migration paths that individuals can take. This topology can be represented as a directed cyclic graph, where nodes are the islands, and the edges are the migration paths. An island model is additionally characterized by the frequency of migrations and the selection and replacement strategies used for individuals on the source and target islands. Figure 1 shows an island model with four islands and a complete graph topology, i.e., individuals can migrate between arbitrary islands.

Using variable-length representations such as graphs is particularly common in GP. The presumably most popular variant of graph-based GP (GGP) is Cartesian GP [38, 39], which was initially developed for evolving two-dimensional electronic circuits. It uses an integer string to encode the structure of a feedforward network (FN), i.e., a directed acyclic graph with dedicated input and output nodes. Another approach for GGP is Evolving Graphs by Graph Programming [4], which uses the graph language GP2 to optimize the structure of FNs. Beyond the domains of GGP, undirected graph representations find application in diverse research areas such as chemical optimization [9, 21] or structural topology optimization [57]. The existing literature on the evolution of directed cyclic graphs is much more limited, which is why new operators have been developed for our use case.

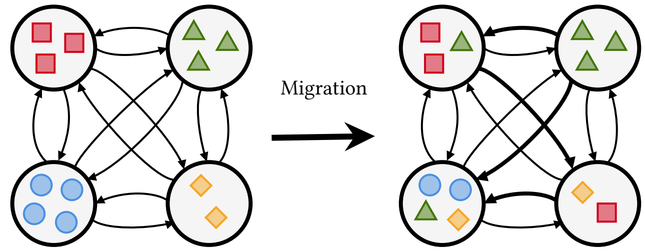


Figure 1: A complete graph topology with four islands.

In general, AAD focuses on finding algorithms that perform well on the problem instances that arise in a given scenario [61]. A formal definition of this search for the optimal algorithm A^* on a distribution $\mathcal{D}$ over the instance space $\mathcal{I}$ can be given as

$$A^* \in \arg\max_{A \in \mathcal{A}} \mathbb{E}[\mathrm{m}(A)|\mathcal{D}] \,, \tag{1}$$

where $\mathcal{A}$ denotes some algorithm space and $\mathrm{m} : \mathcal{A} \times \mathcal{I} \to \mathbb{R}$ is a metric that measures the performance of A on an instance $I \in \mathcal{I}$. In practice, the performance is optimized for a training set of instances $\mathbf{I} \subseteq \mathcal{I}$ that show similar features to the instances likely to be encountered in the future:

$$A^* \in \arg\max_{A \in \mathcal{A}} \mathrm{agg}(\mathrm{m}, A, \mathbf{I}) \,, \tag{2}$$

where agg is a function that aggregates the performance measures on all instances in $\mathbf{I}$, such as the arithmetic mean. The problem in automated algorithm selection lies in finding the algorithm A^* from a set $\mathbf{A} \subseteq \mathcal{A}$ that maximizes m either for a single instance I (per-instance) or an instance set $\mathbf{I}$ (per-set). Parallel algorithm portfolios try to exploit parallelism by executing all algorithms in $\mathbf{A}$ simultaneously on an instance I and returning the best solution found. Alternatively, so-called static and dynamic portfolios allocate computational budgets to algorithms depending on their (expected) performance up-front or dynamically at runtime [23]. Most metaheuristics also have several categorical and numerical parameters that can drastically affect their performance [19]. A specific assignment to these parameters is commonly called a (parameter) configuration, i.e., an element θ from the parameter space Θ. Thus, the goal of automated algorithm configuration is to find the configuration θ^* of the algorithm A_{θ^*} that maximizes performance on $\mathbf{I}$ [48]. In recent years, the line between AAD and automated algorithm configuration has been blurred by representing different algorithm operators as categorical parameters [e.g., 5, 10, 33, 34, 36].

[1] GRAHF is not an acronym.

3 RELATED WORK

While the current literature does not appear to feature comparable approaches to the evolution of heterogeneous island topologies, several approaches share certain characteristics.

For example, several methods investigate the island model from a dynamic point of view. Akhmedova et al. [2] propose a "soft" island model, which features probabilistic migrations, i.e., the genetic operators sample individuals with certain probability from their own island or other islands. In [26], the migration probabilities of a fixed topology are adapted at runtime depending on the impact of the last migration. The approach of Meng et al. maintains high diversity by dynamically reorganizing the island topology with spectral clustering to group similar individuals [37]. In the work of León et al. [27], a hyper-heuristic strategy for heterogeneous multi-objective island models is developed. It distributes the computational budget to islands depending on their hypervolume, allocating more resources to promising algorithms.

Graph representations in AAD are notably limited in the literature, especially representations other than FNs. Shirakawa and Nagao use a directed cyclic graph with common metaheuristic operators as nodes to automatically discover algorithms with multiple cycles [49]. Tisdale et al. encode an evolutionary cycle of metaheuristic operators with a novel directed graph representation [56]. In contrast to [49], nodes may split the individuals that enter through input edges among the output edges, enabling a much more fine-grained view of population-based algorithms. In a recent study, Zhao et al. represent the main loop of a generic metaheuristic algorithm as a directed acyclic graph of operators [62]. Instead of directly manipulating this representation, they utilize a variational graph autoencoder to learn a concise latent embedding, which enables the application of established real-valued optimization methods and surrogate models.

Many approaches apply generative hyper-heuristics to assemble algorithmic components, e.g., using grammar-based GP [40, 55] or Cartesian GP [46, 47]. There are also efforts to evolve optimizers on an instruction level, such as Push GP [29–31]. AAD is furthermore not restricted to metaheuristics as search methodologies, encompassing other fields such as reinforcement learning [28, 60].

4 THE GRAHF FRAMEWORK

GRAHF is a graph-based generative hyper-heuristic framework for evolving heterogeneous island model topologies. It is intended as a high-level approach for hybridizing existing algorithms and adapting them to a given distribution of problem instances, as opposed to designing them from scratch. The framework requires two inputs, namely a training set of problem instances that is representative of the target instance distribution, and a set of algorithms, or heuristics, that can produce solutions for such instances. It also explicitly supports defining parameter spaces for each algorithm, which are optimized together with the overall topology. This means that GRAHF operates completely domain-independent, and can be used "on top" of existing optimization algorithms for any problem where multiple such solvers exist.

In general, searching the space of topologies is supported by three observations: (1) the topology is expected to have a significant impact on the performance of the model [45], (2) no single topology universally outperforms others; instead, its effectiveness is strongly related to the underlying structure of the optimization problem at hand [3], and (3) heterogeneous island models serve as a coarse-grained approach to hybridized metaheuristics, which have been observed to be highly successful [54].

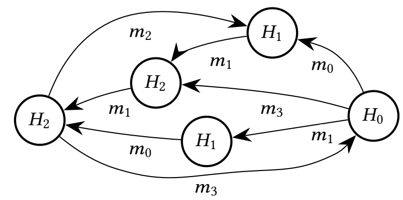


Figure 2: An island model topology with five islands ($\mathbf{H} = \{H_0, H_1, H_2\}$) and eight migration paths ($\mathbf{M} = \{m_0, \ldots, m_3\}$).

The remaining section describes the design space $\mathcal{A}$, the design strategy, i.e., the hyper-heuristic, and the algorithm performance metric m used in GRAHF, loosely following the taxonomy proposed by Zhao et al. [61] for categorizing techniques for AAD.

4.1 Design Space

The topology of a heterogeneous island model algorithm $A \in \mathcal{A}$ is represented as a directed cyclic graph $G = (V, E)$, where V are the islands and E are the paths along which individuals can migrate at fixed intervals. The terms topology and (island) algorithm are used interchangeably since the former fully describes the latter. Each node v and edge e has an associated weight, also called its *type*. For an island, this is a parametrized heuristic H_θ from the set of heuristics $\mathbf{H}$ that operates on the solution space defined by $\mathcal{I}$. The heuristics in $\mathbf{H}$ can range from sophisticated metaheuristics to simple problem-specific heuristic algorithms or special island components such as archives. The edge or migration type $m \in \mathbf{M}$ consists of a migration condition c_{mig}, the operator o_{sel} used to select individuals for migration on the source island, and the operator o_{rep} to replace individuals on the target island, i.e., to integrate the migrated individuals. These conditions and operators may also have parameters, which are grouped into a configuration θ_m for simplicity. The configuration θ_A of a topology A is a set of the configurations θ_{H_v} for each island v and θ_{m_e} for each edge e. Since node and edge types usually have different parameter spaces, each A has its own (but not necessarily unique) composite parameter space Θ_A. An elegant property of this design space is that it is a generalization that includes the heuristics by themselves (singleton graph) and the space of algorithm portfolios (edgeless graph).

Instead of searching Θ_A directly together with $\mathcal{A}$, the design process is modeled as a bilevel optimization problem. The upper-level optimization task consists of finding an optimal topology $A^*_{\theta^*}$, including graph weights, and the lower-level task optimizes Θ_A:

$$A^*_{\theta^*} \in \operatorname*{arg\,max}_{A_{\theta^*} \in \mathcal{A}} \operatorname{agg}(\mathrm{m}, A_{\theta^*}, \mathbf{I}) \tag{3}$$

subject to

$$\theta^* \in \operatorname*{arg\,max}_{\theta \in \Theta_A} \operatorname{agg}(\mathrm{m}, A_\theta, \mathbf{I})\,. \tag{4}$$

Splitting the design space allows the upper-level design strategy to focus on optimizing a graph structure, while the lower-level strategy configures the algorithm. This has the advantage that existing state-of-the-art methods for algorithm configuration can be easily integrated, rather than relying on one algorithm to solve both tasks simultaneously, which may result in one task being neglected in favor of the other. A disadvantage is that this leads to a nested search, where the algorithm configurator is invoked many times during the upper-level optimization, possibly requiring more computational budget. This clear division of tasks is expected to provide more potential for improvement compared to a flattened search space.

4.2 Topology Design Strategy: EDGE

Topology design and algorithm configuration are considered separate tasks by GRAHF and are therefore addressed by two different optimizers, the latter of which can be essentially any existing algorithm configuration method. For the former, we propose a new population-based GGP algorithm for evolving directed cyclic graphs with categorical node and edge weights, called naivE Directed Graph Evolution (EDGE). It applies the standard evolutionary loop of selection, probabilistic crossover, mutation, and replacement.

EDGE[2] uses tournament selection with a small tournament size and generational (or μ, λ) replacement, as it is usually instantiated with a small to medium-sized population. The algorithm maintains an archive of the n_{ar} best individuals during the search and inserts a random elitist from the archive into the population every f_{eli} iterations. While the replacement serves to enhance exploration capabilities, the archiving mechanism exploits the neighborhood of known elitists, which undergo crossover and mutation in the next iteration. The elitist might also be left unmodified in the population, which effectively acts as a restart of the configuration mechanism, leading to a re-evaluation of the elitist topology with a different (locally optimal) parameter set.

Initialization. The population of EDGE is initialized with $n_{\mathrm{pop}}+m$ individuals, where n_{pop} individuals are randomly generated. The number of nodes is sampled uniformly within the interval $[2, n_{\mathrm{init}}]$, and edges are added with probability p_e, resulting in denser or sparser graphs. The node and edge types are uniformly sampled from **H** and **M**. The remaining m individuals consist of a mixture of established topologies such as star, wheel, and complete graphs, in addition to singleton graphs of those heuristics in **H** that might be sufficient on their own. In combination with the elitist archive, this guarantees that every search yields a topology that is better than or equal to the best heuristic in **H**.

Crossover. The crossover operator combines two parent graphs into two offspring graphs. It involves finding a cut for each parent graph that separates it into two non-empty partitions. Then, the first partition of the first parent is merged with the first partition of the second parent to create the first child; the second child is created from the other two partitions. The outgoing edges of each partition that were cut are reconnected to random nodes in the other partition, connecting them in both directions. Note that this operator preserves the total number of nodes and edges between parents and offspring.

[2] Detailed pseudocode of EDGE is given in the supplementary material.

The partitions are obtained by sampling the size of the partitions n_{split} from a binomial distribution, which biases the cut towards an even split while still allowing uneven splits. The first n_{split} nodes are chosen for the first partition, and the remaining nodes for the second, while ensuring that each partition contains at least a single node. Naturally, arbitrary methods for graph partitioning can be applied here, e.g., minimizing the number of edge cuts.

Mutation. Mutation in EDGE consists of four sub-operators, i.e., node mutation, edge mutation, node weight resampling, and edge weight resampling, all of which are invoked depending on their mutation rate. Node mutation either adds a new random node to the graph and connects it to other random nodes in both directions or removes a random node along with connected edges. Edges are mutated by adding random non-existing edges or removing random existing edges with equal probability. The node and edge types are resampled occasionally, choosing from **H** and **M** with uniform probability. Note that all mutation sub-operators guarantee that the mutated graphs are non-empty. While there is no upper bound on the number of nodes that can be inserted by mutation, unrestricted growth of graph size has not been observed, possibly because only single-node mutations are used.

4.3 Algorithm Performance Metric

Creating a robust metric to assess algorithm performance across multiple problem instances poses a non-trivial challenge. The stochastic nature of metaheuristics often leads to significant variability in performance [54], and differences between instances, such as variations in the magnitude of objective values and overall difficulty, further complicate the evaluation process.

One approach to address these complexities is the fixed-target approach, measuring algorithm *runtime* [13]. Unlike the fixed-budget approach, which sets a maximum evaluation budget and reports the minimum objective value achieved, the fixed-target approach sets a predetermined objective value and reports the number of evaluations or runtime needed. Defining such a target value is challenging for less-studied instances, but using runtime allows for unbiased quantitative algorithm comparisons [14]. The fixed-budget approach has the disadvantage that it treats early convergence to a good solution identically to exhausting the entire budget, while the fixed-target approach conversely excludes runs close to the target but not reaching it. In general, this is less of a drawback because the acceptability of such runs largely depends on the objective function. The fixed-target approach also requires an evaluation budget in practice, since the algorithm may not reach the target at all.

In this paper, we adopt the definition and notation for runtime (RT) established by Hansen et al. in [14] and similarly call a run *successful* if the algorithm reaches the target value, and *unsuccessful* if it does not. The expected runtime (ERT) is then an estimate of $\mathbb{E}[\mathrm{RT}]$ on a given data set consisting of n_{s} successful runs with runtimes $\mathrm{RT}_i^{\mathrm{s}}$ and n_{us} unsuccessful runs with evaluations $\mathrm{EV}_j^{\mathrm{us}}$, and can be reduced to the total number of function evaluations by both successful and unsuccessful runs (#evals) divided by the number of successful runs:

$$\mathrm{ERT}(A, I) = \frac{1}{n_{\mathrm{s}}} \sum_i \mathrm{RT}_i^{\mathrm{s}} + \frac{1-p_{\mathrm{s}}}{p_{\mathrm{s}}} \frac{1}{n_{\mathrm{us}}} \sum_j \mathrm{EV}_j^{\mathrm{us}} = \frac{\#\mathrm{evals}}{n_{\mathrm{s}}}\,. \quad (5)$$

If all runs are unsuccessful, a penalty value is obtained by setting n_s to 0.01, i.e., multiplying the total number of evaluations by 100.

Running the same algorithm on instances of different difficulty usually results in ERTs with different ranges, so simply averaging over instances can cause a small set of very difficult instances to dominate runtime. While this is not necessarily undesirable behavior, GRAHF also supports weighting the instances in $\mathbf{I}$ based on importance. To allow consistent and comparable instance weights, the runtime can be standardized across instances. The so-called median-standardized ERT (MERT) of A on I is written as

$$\mathrm{MERT}(A, I) = \frac{\mathrm{ERT}(A, I) - \tilde{\mu}_I}{\tilde{\sigma}_I}\,, \text{ where}$$
$$\tilde{\mu}_I = \operatorname*{median}_{A' \in \mathcal{A}} \mathrm{ERT}(A', I)\,,\ \tilde{\sigma}_I = \operatorname*{median}_{A' \in \mathcal{A}} |\,\mathrm{ERT}(A', I) - \tilde{\mu}_I|\,, \tag{6}$$

i.e., the ERT of A is standardized using the median and median average deviation of the ERTs of other algorithms $A' \in \mathcal{A}$, which act as robust measures for difficulty and variability of runtime on an instance. A positive MERT indicates that an algorithm performs worse than the median while a negative value indicates better performance, with the magnitude of the value showing the degree of deviation. Calculating the median and median average deviation ERT over all possible algorithms is not feasible in practice, so GRAHF approximates this value using the performance values from previous iterations. Since the approximations of $\tilde{\mu}_I$ and $\tilde{\sigma}_I$ are dynamic, the MERT of an individual must be re-evaluated after an iteration using the cached ERT. Note that this re-evaluation does not change the ranking of algorithms. Finally, the aggregated performance measure of an algorithm A on a training set of instances $\mathbf{I}$ is the weighted sum of the MERT of all $I \in \mathbf{I}$:

$$\mathrm{agg}(\mathrm{MERT}, A, \mathbf{I}) = \frac{\sum_{I \in \mathbf{I}} w_I \,\mathrm{MERT}(A, I)}{\sum_{I \in \mathbf{I}} w_I}\,. \tag{7}$$

5 TRAINING EXPERIMENTS

GRAHF is investigated on 24 training sets composed of real-valued benchmark functions with different instance features and dimensionalities, resembling what is commonly called "training" in machine learning contexts. While these benchmark functions have received criticism for their limited correlation to problems encountered in real-world settings [e.g. 11], they offer distinct advantages over other problem types. Specifically, their fast execution makes them particularly suitable for AAD, and they are exceptionally well studied, meaning that instance properties such as the objective value of the global optimum are readily available. In addition, there exists a wide array of benchmark functions with diverse landscape features, and it is possible to generate variants through transformations such as translation, rotation, and scaling.

5.1 Training Sets

We consider two different types of benchmark functions for training. The first 18 training sets are composed of a total of 55 real-valued functions commonly used as benchmarks [7, 20, 52], each set reflecting a different combination of instance features to simulate similar instances from some distribution. These features include dimensionality (fixed or arbitrary up to 10D), convexity, modality, separability, and observed difficulty in preliminary experiments ("easier" or "harder"). Specifically, training sets 1 to 6 investigate the impact of difficulty, while sets 7 to 11 examine both different dimensionalities and difficulties. Notably, sets 7, 9, 10, and 11 encompass the same functions in various dimensionalities. Sets 12 to 15 explore diverse combinations of instance features in low dimensions, whereas sets 16 to 18 focus on higher dimensionality.

Training sets 19 to 24 are sourced from the well-known COCO BBOB benchmark suite [15]. The first five training sets represent each BBOB subgroup, namely separable (F1), moderate (F2), ill-conditioned (F3), multimodal globally structured (F4), and multimodal with weak structure (F5), each function present in five different variants through pseudo-random transformations (in COCO terms, "instances"). The last set contains all functions in a simple design to achieve a balanced set size. Notably, the first two training sets are used in higher dimensionality (5D) than the others (2D), considering that these instances are generally easier to solve.

Detailed tables listing all benchmark functions, the instance features, and exact training set compositions for all experiments can be found in the supplementary material.

5.2 Graph Primitives

Heuristics. Five heuristics are selected for the experiments, namely Particle Swarm Optimization (PSO) [22, 42], Differential Evolution (DE) [50], a non-adaptive evolution strategy (ES) [8], Simulated Annealing (SA) [24], and a local search (LS) [18]. Each heuristic exposes a number of parameters and components that are tuned by the configurator, a detailed overview of which can be found in the supplementary material. In addition to these well-established approaches for real-valued optimization, two domain-independent archive primitives are considered. The primitive called *archive* simply initializes its population with random solutions at the start of the search, and performs no modifications thereafter. It effectively serves as a repository for promising individuals from other islands, acting as a stopover before later migrating to other islands or back to the origin island. The *resampling archive* introduces diversity into the process by uniformly resampling a portion of its population in the search space, potentially re-initializing other islands through migration. This heuristic set is designed to include a wide variety of algorithms, incorporating both sophisticated methods and simpler approaches. The metaheuristics also expose a parameter that controls the number of so-called *sub-iterations*. The migration of individuals occurs at fixed intervals, with this interval between successive migrations being referred to as an *iteration* of the island model algorithm. Instead of allocating an equal computational budget to each heuristic within an iteration, the number of sub-iterations per global iteration is also tuned by the configuration mechanism. This effectively acts as a weight for an island, as heuristics with more sub-iterations exert more influence on the optimization process.

Migrations. Each edge has a specific migration condition and operators for selecting and replacing (integrating) individuals. For the experiments in this paper, only a probabilistic migration condition is employed; this consequently means that iterations are not guaranteed to witness any migrations. Both the random and tournament selection operators select a percentage of individuals from the population of the source island. The tournament selection further utilizes a tournament size ratio, dictating the size of the tournaments conducted relative to the number of selected individuals. The $(\mu + \lambda)$ replacement operator discards the worst individuals to allow for room to integrate the migrated individuals, while the other replacement chooses them randomly. Both operators maintain the population size of the target island. In total, one migration condition, along with two possible selection and replacement operators, results in four distinct migration types.

5.3 Experimental Setup

As described in Section 4.2, the initial population of EDGE is extended with $m = n_{\text{pop}}$ PSO, DE, ES, and SA singleton graphs in addition to randomly generated star, wheel, and complete topologies. The exact parameter values used for EDGE are given in the supplementary material.

Iterated racing, or more precisely the `irace` package [32] is used as a state-of-the-art approach for algorithm configuration, i.e., for solving the lower-level optimization task. Because the racing mechanism requires a performance measure of a configuration on a single instance, `irace` optimizes the ERT of $n_{\text{crep}} = 10$ executions. A computational budget of at least $n_{\text{irace}} = 300$ configuration evaluations is given for each execution of `irace`.

After `irace` has identified (locally) optimal parameters for an algorithm, the island model algorithm is executed $n_{\text{rep}} = 60$ times on all instances in the evaluation phase. The target value for an instance is set to the objective value of the known global optimum, plus a small offset ϵ (10^{-6} by default). The maximal budget of function evaluations EV^{us} for one run is set to 2×10^5. Uniform weights are assigned to all functions within a training set. Some training sets use different values for performance measure parameters to provide initial insights into their effects (e.g., higher tuning budget); more information can be found in the supplementary material.

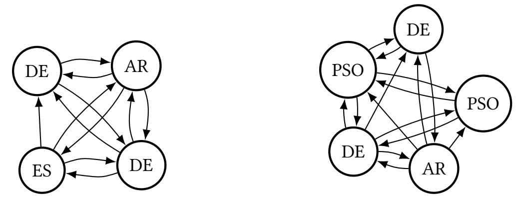
Note that a single run of GRAHF is a significant computational effort (here, ca. 600 CPU hours per training set), since $n_{\text{pop}} \cdot n_{\text{iter}} \cdot (n_{\text{irace}} \cdot n_{\text{crep}} + n_{\text{rep}})$ algorithms are evaluated. A key factor for efficiency is therefore not only that the population of EDGE can be tuned and evaluated in parallel, but also that each island model topology allows parallel execution of its heuristic nodes. It should be noted, however, that despite the high degree of parallelization, the significant runtime remains a challenge, which is why the parameter values of GRAHF were mainly determined by intuition.

The core functionality of GRAHF is implemented in Rust[3] using the MAHF framework[4] [59], with `irace` being called through `irace-rs`[5]. While the common benchmark functions are implemented in pure Rust, the COCO BBOB suite is integrated using `coco-accelerated`[6], a hardware-accelerated reimplementation.

5.4 Results and Discussion

This section is dedicated to examining both the training process and the resulting elitist topologies. Particular focus will be placed on the analysis of training set 18, consisting of twenty-four ten-dimensional benchmark functions, and therefore being a comparatively high-dimensional example. We denote A_i^j as the elitist topology j from the final archive in the run on training set i.

[3] https://doi.org/10.5281/zenodo.10606197 [4] https://github.com/mahf-opt/mahf
[5] https://github.com/saethox/irace-rs [6] https://github.com/luleyleo/coco-accelerated



(a) A_{18}^0 **(b)** A_{18}^1

Figure 3: The two best elitist island model topologies on twenty-four 10D instances (training set 18).

The training demonstrates that GRAHF is able to generate a variety of topologies for different training sets, from very large and dense graphs to small sparse graphs, including singleton graphs. Figure 3 shows the two best topologies found for training set 18, omitting the migration types for clarity, with AR denoting the simple archive. Both A_{18}^0 and A_{18}^1 are nearly complete graph topologies with four and five nodes, respectively. The former contains two DE, one ES, and one archive, with one DE having the highest weight by a wide margin. The latter is composed of two PSOs and two DEs with relatively equal weights, and one archive.

All elitist topologies contain at least one island with either PSO or DE, which is expected since they are the most sophisticated metaheuristics considered here. For example, PSO singletons (or with archives) show very high performance on training sets 1, 13, 14, 15, 23, and 24, reinforcing the reputation of PSO as a robust and powerful metaheuristic. Notably, DE is selected more frequently on training sets with many higher-dimensional separable functions (e.g., training sets 16, 17, and 18), which is consistent with previous results [e.g. 44, 53]. SA mostly plays a supporting role, often appearing in conjunction with either PSO and DE and (multiple) archives. Both ES and LS make only a few appearances, with ES being more common than LS. The archive can be found in almost all topologies, most likely because an additional archive consumes only a negligible amount of the evaluation budget, but potentially allows for much more complex population dynamics. The resampling archive is sparsely selected, which may indicate that diversity is generally not a limiting factor.

Figure 4 shows the optimization process on training set 18. The left plot shows the MERT distribution of the population per iteration, marking a true improvement over the previous best topology with a green cross. Note that this is not a convergence plot; MERT is a relative measure, and therefore only comparable in the same iteration. It can be observed that the last improving elitist topology, i.e., A_{18}^0, is generated in iteration 51. It shows negative MERT, which means that it is better than the median topology on all instances.

The diversity of the population is given in the plot on the right, where an approximation of the graph edit distance [1] is used as a metric. The figure shows the distribution of all pairwise comparisons, or pairwise graph edit distance (PGED), ignoring outliers, and marking the median with a line. While the diversity seems to decrease with iterations in most runs, there are also some cases where peaks can be observed (e.g., iteration 48 in Figure 4). Sometimes even several peaks appear, or the diversity increases with iterations. This is most likely due to the (μ, λ) selection since it

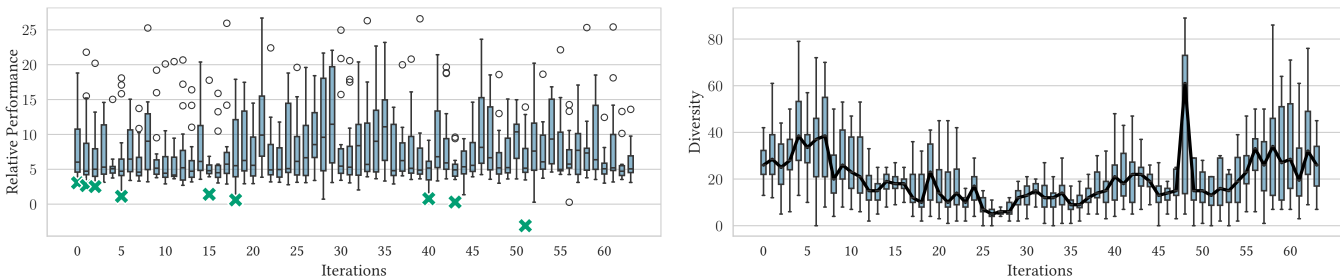


Figure 4: MERT (left) and PGED (right) per iteration on twenty-four 10D instances (training set 18).

allows unconstrained evolution, and the crossover operator, which can merge two medium-sized topologies into one large topology.

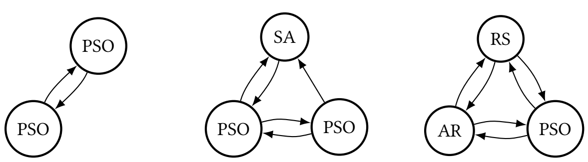
There are also some benchmark functions where not a single algorithm was able to hit the required target value. This effectively causes the instance to be ignored, as all algorithms receive a similar penalty. Examples of this are two problem instances in training set 18; this penalty factor has been eliminated in Figure 4 for clarity. A possible explanation for this could be incorrect target values for some instances, which calls for a closer investigation.

Another interesting observation is that GRAHF exhibits considerable variance in ERT across distinct training sets, with notable differences in medians. For example, on the sphere function, the median ERT is 1183 for training set 12, but 2605 for training set 14, suggesting divergent paths through topology space. Note that this median is used only for standardization, and the distributions still agree on relative algorithm performance.

In general, the runs with a higher tuning budget and more evaluation iterations showed less variation in relative performance, suggesting that more computational budget is worthwhile.

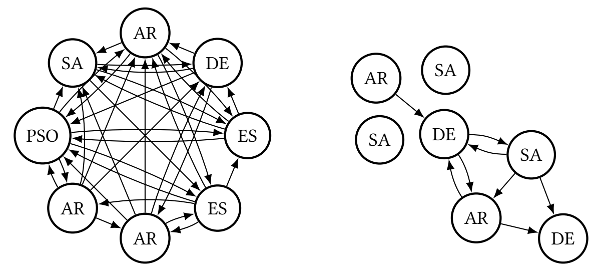
6 TOPOLOGY POST-EVALUATION

In the post-evaluation, we compare a selection of diverse elitist topologies on three instance sets, two of which are extensions of the training sets, i.e., the common benchmark functions (BMF) and the COCO BBOB suite. The COCO and BMF instances that can be instantiated in arbitrary dimensionality are used in 2, 3, 5, 10, and 20 dimensions, resulting in a total of 120 and 151 instances, respectively. The COCO set employs transformed instances that were technically not seen during training. The last set is composed of 22 real-world engineering optimization problems (ENOP) taken from [25], 14 of which arise in industrial chemical processes and 8 in mechanical design. These problems are characterized by their high complexity, i.e., being non-convex and having many (in)equality



(a) A_2^0 **(b)** A_6^0 **(c)** A_9^0

Figure 5: Three top-ranking topologies on BMF.



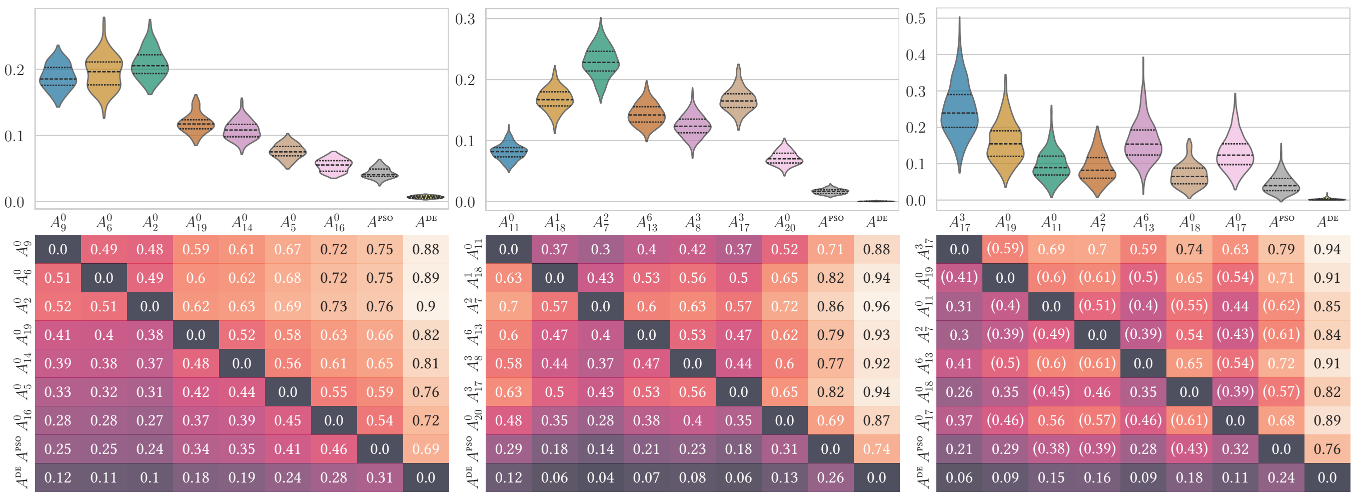
(a) A_7^2 **(b)** A_{17}^3

Figure 6: Top-ranking topologies on COCO (a) and ENOP (b).

constraints. Given that the generated algorithms lack a dedicated mechanism for handling constraints, the problems are transformed into unconstrained problems using a penalty function.

The algorithms considered here include 34 either top-performing or interesting elitist island model topologies and two baselines, i.e., representative metaheuristic primitives on their own. Specifically, we use PSO and DE, subsequently denoted as A^{PSO} and A^{DE}, with common parameter values [42, 50] to contextualize the results obtained here. Each of the algorithms is executed 300 times on each instance of the BMF and COCO sets, and 30 times on the ENOP instances, which are more computationally expensive to evaluate.

We use a Bayesian performance analysis [43] to interpret the results. This approach requires a ranking of algorithms on each instance, which is obtained as follows: first, algorithms with successful runs are ranked against each other using the ERT, with (two arisen) ties being resolved by choosing a random winner. Algorithms that fail to reach the target value on an instance are ranked using the mean of the best objective value found over all repetitions. These two rankings are then combined into a global ranking that places algorithms ranked by objective value behind any algorithm ranked by ERT, thereby strongly preferring fully solved instances over mean performance rankings. On the ENOP instances, algorithms are ranked only by the mean best objective value, since no target is available. Six ENOP instances were excluded from the ranking because all pre-selected algorithms performed identically. The posterior distribution of the Bayesian analysis can then be transformed into more informative posterior distributions, such as the probability of an algorithm A_i being ranked the first (top-ranking) and of algorithm A_i outperforming algorithm A_j (pairwise outranking). Since the analysis has a complexity of $O(n!)$, we pre-select



(a) BMF **(b) COCO** **(c) ENOP**

Figure 7: Top-ranking (top) and pairwise outranking (bottom) probabilities using Bayesian performance analysis.

the seven top-performing algorithms using the median and mean ranks over all instances in a training set (plus both baselines).

The results are visualized in Figure 7, with the top-ranking probabilities given as violin plots, which include a measure of uncertainty. The pairwise outranking probabilities are displayed as heatmaps, with only ENOP having values with a variance >0.01 (but <0.02), which are shown in parentheses.

In Figure 7a, it can be observed that A_9^0, A_6^0, and A_2^0 have a substantially higher probability of being ranked first on the BMF instances; the pairwise outranking probabilities show that there is no significant probability of these three algorithms outranking each other. On the other hand, all three topologies have a higher than 75 % probability of outperforming A^{PSO} and almost 90 % for A^{DE}. All three algorithms are small and dense (cf. Figure 5), with A_2^0 being a two-island PSO ring, A_6^0 being an extension of the former with one SA island, and A_9^0 being a complete graph with a PSO, an archive (AR), and a resampling archive (RS).

On COCO, it is worth noting that the rank pre-selection resulted in completely different elitist sets. Figure 7b shows that A_7^2 has the highest probability of outranking all other algorithms; the pairwise outranking probabilities reveal that it outperforms its closest competitors, A_{18}^1 and A_{17}^3, with 57 % probability, which is only of limited significance. Overall, A_7^2 beats four of the remaining algorithms with a probability greater than 70 %. All selected topologies have a very high probability of beating both baselines in the ranking, with A^{PSO} doing better than A^{DE}. Notably, A_7^2 is a very large, almost complete graph, which is topologically very different from the elitists on BMF. It is given in Figure 6a.

For ENOP, some overlap in the pre-selected elitists is apparent, with A_{19}^0 being present in the BMF set and A_{11}^0, A_{13}^6, A_{17}^3, and A_7^2 being shared with COCO. A_{17}^3 has the highest probability of being ranked first, and has at least 60 % probability of beating any other algorithm. Its closest competitors, albeit with some margin, are A_{19}^0, A_{13}^6, both of which do not employ a single PSO island, and A_{17}^0. The topology of A_{17}^3 is given in Figure 6b, with two unconnected SA islands having high weights, accompanied by two DEs with small population sizes. It is worth noting that experiments 13 and 17 only include non-convex instances, which coincides with the ENOP instances; however, experiment 19 only includes very easy instances, underlining that only limited parallels can be drawn here. A^{PSO} and A^{DE} are again significantly outperformed by A_{17}^3.

Overall, this evaluation has determined not only the effectiveness of the evolved hybrids across multiple optimization scenarios but also examined the generalization abilities of specific elitists, and verified the significant influence of topology on performance.

7 CONCLUSION AND FUTURE WORK

This article has introduced GRAHF as an innovative hyper-heuristic framework for optimizing the topology and parameters of heterogeneous island model metaheuristics. The framework was evaluated on 24 training sets with a wide range of landscape features, selecting optimal heuristics and migrations for each set. A post-evaluation of 34 selected elitists on extensive benchmarks and real-world instances revealed remarkable generalization abilities, often significantly outperforming non-island baseline algorithms. These results have identified great potential in GRAHF, but several key areas deserve further attention. One specific aspect is the integration of self-adaptive mechanisms into GRAHF to dynamically control its parameters since statically tuning is extremely expensive. The use of surrogates on the graph representation could also significantly reduce runtime. Another promising consideration is to integrate statistical tools into the algorithm performance measure, e.g., guiding the search process by automated Bayesian analysis. The search space itself could be extended to dynamic topologies and appropriate adaptation strategies based on runtime information. Analyzing GRAHF on correlated real-world training sets using state-of-the-art metaheuristics (e.g., CMA-ES [16]) as primitives is an additional key point for future work. In summary, GRAHF is a promising approach for automating the design of coarse-grained hybrid metaheuristics.

REFERENCES

[1] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. 2015. An Exact Graph Edit Distance Algorithm for Solving Pattern Recognition Problems. In *4th International Conference on Pattern Recognition Applications and Methods 2015*. Lisbon, Portugal. https://doi.org/10.5220/0005209202710278

[2] Shakhnaz Akhmedova, Vladimir Stanovov, and Eugene Semenkin. 2018. Soft Island Model for Population-Based Optimization Algorithms. In *Advances in Swarm Intelligence (Lecture Notes in Computer Science)*, Ying Tan, Yuhui Shi, and Qirong Tang (Eds.). Springer International Publishing, Cham, 68–77. https://doi.org/10.1007/978-3-319-93815-8_8

[3] Ignacio Arnaldo, Iván Contreras, David Millán-Ruiz, J. Ignacio Hidalgo, and Natalio Krasnogor. 2013. Matching Island Topologies to Problem Structure in Parallel Evolutionary Algorithms. *Soft Computing* 17, 7 (July 2013), 1209–1225. https://doi.org/10.1007/s00500-013-0995-9

[4] Timothy Atkinson, Detlef Plump, and Susan Stepney. 2018. Evolving Graphs by Graph Programming. In *Genetic Programming (Lecture Notes in Computer Science)*, Mauro Castelli, Lukas Sekanina, Mengjie Zhang, Stefano Cagnoni, and Pablo García-Sánchez (Eds.). Springer International Publishing, Cham, 35–51. https://doi.org/10.1007/978-3-319-77553-1_3

[5] Doğan Aydın, Gürcan Yavuz, and Thomas Stützle. 2017. ABC-X: A Generalized, Automatically Configurable Artificial Bee Colony Framework. *Swarm Intelligence* 11, 1 (March 2017), 1–38. https://doi.org/10.1007/s11721-017-0131-z

[6] Theodore C. Belding. 1995. The Distributed Genetic Algorithm Revisited. https://doi.org/10.48550/arXiv.adap-org/9504007 arXiv:adap-org/9504007

[7] Hans-Georg Beyer and Steffen Finck. 2012. HappyCat – A Simple Function Class Where Well-Known Direct Search Algorithms Do Fail. In *Parallel Problem Solving from Nature - PPSN XII (Lecture Notes in Computer Science)*, Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone (Eds.). Springer, Berlin, Heidelberg, 367–376. https://doi.org/10.1007/978-3-642-32937-1_37

[8] Hans-Georg Beyer and Hans-Paul Schwefel. 2002. Evolution Strategies – A Comprehensive Introduction. *Natural Computing* 1, 1 (March 2002), 3–52. https://doi.org/10.1023/a:1015059928466

[9] Nathan Brown, Ben McKay, François Gilardoni, and Johann Gasteiger. 2004. A Graph-Based Genetic Algorithm and Its Application to the Multiobjective Evolution of Median Molecules. *Journal of Chemical Information and Computer Sciences* 44, 3 (May 2004), 1079–1087. https://doi.org/10.1021/ci034290p

[10] Christian L. Camacho-Villalón, Marco Dorigo, and Thomas Stützle. 2022. PSO-X: A Component-Based Framework for the Automatic Design of Particle Swarm Optimization Algorithms. *IEEE Transactions on Evolutionary Computation* 26, 3 (June 2022), 402–416. https://doi.org/10.1109/tevc.2021.3102863

[11] Andreas Fischbach and Thomas Bartz-Beielstein. 2020. Improving the Reliability of Test Functions Generators. *Applied Soft Computing* 92 (July 2020), 106315. https://doi.org/10.1016/j.asoc.2020.106315

[12] Michel Gendreau and Jean-Yves Potvin. 2018. *Handbook of Metaheuristics* (3rd ed. 2019 edition ed.). Springer, New York, NY.

[13] Nikolaus Hansen, Anne Auger, Dimo Brockhoff, Dejan Tušar, and Tea Tušar. 2016. COCO: Performance Assessment. https://doi.org/10.48550/arXiv.1605.03560 arXiv:1605.03560 [cs]

[14] Nikolaus Hansen, Anne Auger, Dimo Brockhoff, and Tea Tušar. 2022. Anytime Performance Assessment in Blackbox Optimization Benchmarking. *IEEE Transactions on Evolutionary Computation* 26, 6 (Dec. 2022), 1293–1305. https://doi.org/10.1109/tevc.2022.3210897

[15] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. 2021. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *Optimization Methods and Software* 36, 1 (Jan. 2021), 114–144. https://doi.org/10.1080/10556788.2020.1808977

[16] Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. 2003. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation* 11, 1 (March 2003), 1–18. https://doi.org/10.1162/106365603321828970

[17] Sean Harris, Travis Bueter, and Daniel R. Tauritz. 2015. A Comparison of Genetic Programming Variants for Hyper-Heuristics. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO Companion '15)*. Association for Computing Machinery, New York, NY, USA, 1043–1050. https://doi.org/10.1145/2739482.2768456

[18] Holger Hoos and Thomas Stützle. 2004. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[19] Changwu Huang, Yuanxiang Li, and Xin Yao. 2020. A Survey of Automatic Parameter Tuning Methods for Metaheuristics. *IEEE Transactions on Evolutionary Computation* 24, 2 (April 2020), 201–216. https://doi.org/10.1109/tevc.2019.2921598

[20] Momin Jamil and Xin-She Yang. 2013. A Literature Survey of Benchmark Functions For Global Optimization Problems. *International Journal of Mathematical Modelling and Numerical Optimisation* 4, 2 (2013), 150. https://doi.org/10.1504/ijmmno.2013.055204 arXiv:1308.4008 [cs, math]

[21] Jan Jensen. 2019. A Graph-Based Genetic Algorithm and Generative Model/Monte Carlo Tree Search for the Exploration of Chemical Space. *Chemical Science* 10, 12 (2019), 3567–3572. https://doi.org/10.1039/C8SC05372C

[22] James Kennedy and Russell Eberhart. 1995. Particle Swarm Optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, Vol. 4. 1942–1948 vol.4. https://doi.org/10.1109/icnn.1995.488968

[23] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. 2019. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation* 27, 1 (March 2019), 3–45. https://doi.org/10.1162/evco_a_00242

[24] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220, 4598 (May 1983), 671–680. https://doi.org/10.1126/science.220.4598.671

[25] Abhishek Kumar, Guohua Wu, Mostafa Z. Ali, Rammohan Mallipeddi, Ponnuthurai Nagaratnam Suganthan, and Swagatam Das. 2020. A Test-Suite of Non-Convex Constrained Optimization Problems from the Real-World and Some Baseline Results. *Swarm and Evolutionary Computation* 56 (Aug. 2020), 100693. https://doi.org/10.1016/j.swevo.2020.100693

[26] Frédéric Lardeux and Adrien Goëffon. 2010. A Dynamic Island-Based Genetic Algorithms Framework. In *Simulated Evolution and Learning (Lecture Notes in Computer Science)*, Kalyanmoy Deb, Arnab Bhattacharya, Nirupam Chakraborti, Partha Chakroborty, Swagatam Das, Joydeep Dutta, Santosh K. Gupta, Ashu Jain, Varun Aggarwal, Jürgen Branke, Sushil J. Louis, and Kay Chen Tan (Eds.). Springer, Berlin, Heidelberg, 156–165. https://doi.org/10.1007/978-3-642-17298-4_16

[27] Coromoto León, Gara Miranda, Eduardo Segredo, and Carlos Segura. 2009. Parallel Hypervolume-Guided Hyperheuristic for Adapting the Multi-objective Evolutionary Island Model. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2008)*, Natalio Krasnogor, María Belén Melián-Batista, José Andrés Moreno Pérez, J. Marcos Moreno-Vega, and David Alejandro Pelta (Eds.). Springer, Berlin, Heidelberg, 261–272.

[28] Ke Li and Jitendra Malik. 2016. Learning to Optimize. https://doi.org/10.48550/arXiv.1606.01885 arXiv:1606.01885 [cs, math, stat]

[29] Michael A. Lones. 2019. Instruction-Level Design of Local Optimisers Using Push GP. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '19)*. Association for Computing Machinery, New York, NY, USA, 1487–1494. https://doi.org/10.1145/3319619.3326806

[30] Michael A. Lones. 2020. Optimising Optimisers with Push GP. In *Genetic Programming (Lecture Notes in Computer Science)*, Ting Hu, Nuno Lourenço, Eric Medvet, and Federico Divina (Eds.). Springer International Publishing, Cham, 101–117. https://doi.org/10.1007/978-3-030-44094-7_7

[31] Michael A. Lones. 2021. Evolving Continuous Optimisers from Scratch. *Genetic Programming and Evolvable Machines* 22, 4 (Dec. 2021), 395–428. https://doi.org/10.1007/s10710-021-09414-8

[32] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. 2016. The Irace Package: Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives* 3 (Jan. 2016), 43–58. https://doi.org/10.1016/j.orp.2016.09.002

[33] Manuel López-Ibánez, Marie-Eléonore Kessaci, and Thomas Stützle. 2017. *Automatic Design of Hybrid Metaheuristics from Algorithmic Components*. Technical Report. IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

[34] Manuel Lopez-Ibanez and Thomas Stützle. 2012. The Automatic Design of Multiobjective Ant Colony Optimization Algorithms. *IEEE Transactions on Evolutionary Computation* 16, 6 (Dec. 2012), 861–875. https://doi.org/10.1109/tevc.2011.2182651

[35] Gabriel Luque and Enrique Alba. 2011. *Parallel Genetic Algorithms*. Studies in Computational Intelligence, Vol. 367. Springer, Berlin, Heidelberg.

[36] Marie-Eléonore Marmion, Franco Mascia, Manuel López-Ibáñez, and Thomas Stützle. 2013. Automatic Design of Hybrid Stochastic Local Search Algorithms. In *Hybrid Metaheuristics (Lecture Notes in Computer Science)*, María J. Blesa, Christian Blum, Paola Festa, Andrea Roli, and Michael Sampels (Eds.). Springer, Berlin, Heidelberg, 144–158. https://doi.org/10.1007/978-3-642-38516-2_12

[37] Qinxue Meng, Jia Wu, John Ellis, and Paul J. Kennedy. 2017. Dynamic Island Model Based on Spectral Clustering in Genetic Algorithm. In *2017 International Joint Conference on Neural Networks (IJCNN)*. 1724–1731. https://doi.org/10.1109/IJCNN.2017.7966059

[38] Julian F. Miller (Ed.). 2011. *Cartesian Genetic Programming*. Springer, Berlin, Heidelberg.

[39] Julian Francis Miller. 2020. Cartesian Genetic Programming: Its Status and Future. *Genetic Programming and Evolvable Machines* 21, 1 (June 2020), 129–168. https://doi.org/10.1007/s10710-019-09360-6

[40] Péricles B. C. Miranda and Ricardo B. C. Prudêncio. 2020. A Novel Context-Free Grammar for the Generation of PSO Algorithms. *Natural Computing* 19, 3 (Sept. 2020), 495–513. https://doi.org/10.1007/s11047-018-9679-9

[41] Riccardo Poli and Mario Graff. 2009. There Is a Free Lunch for Hyper-Heuristics, Genetic Programming and Computer Scientists. In *Genetic Programming (Lecture Notes in Computer Science)*, Leonardo Vanneschi, Steven Gustafson, Alberto Moraglio, Ivanoe De Falco, and Marc Ebner (Eds.). Springer, Berlin, Heidelberg, 195–207. https://doi.org/10.1007/978-3-642-01181-8_17

[42] Riccardo Poli, James Kennedy, and Tim Blackwell. 2007. Particle Swarm Optimization. *Swarm Intelligence* 1, 1 (June 2007), 33–57. https://doi.org/10.1007/s11721-007-0002-0

[43] Jairo Rojas-Delgado, Josu Ceberio, Borja Calvo, and Jose A. Lozano. 2022. Bayesian Performance Analysis for Algorithm Ranking Comparison. *IEEE Transactions on Evolutionary Computation* 26, 6 (Dec. 2022), 1281–1292. https://doi.org/10.1109/TEVC.2022.3208110

[44] Jani Rönkkönen, Saku Kukkonen, and Kenneth V. Price. 2005. Real-Parameter Optimization with Differential Evolution. In *2005 IEEE Congress on Evolutionary Computation*, Vol. 1. 506–513 Vol.1. https://doi.org/10.1109/CEC.2005.1554725

[45] M. Ruciński, D. Izzo, and F. Biscani. 2010. On the Impact of the Migration Topology on the Island Model. *Parallel Comput.* 36, 10 (Oct. 2010), 555–571. https://doi.org/10.1016/j.parco.2010.04.002

[46] Patricia Ryser-Welch, Julian F. Miller, and Shahriar Asta. 2015. Generating Human-readable Algorithms for the Travelling Salesman Problem Using Hyper-Heuristics. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO Companion '15)*. Association for Computing Machinery, New York, NY, USA, 1067–1074. https://doi.org/10.1145/2739482.2768459

[47] Patricia Ryser-Welch, Julian F. Miller, Jerry Swan, and Martin A. Trefzer. 2016. Iterative Cartesian Genetic Programming: Creating General Algorithms for Solving Travelling Salesman Problems. In *Genetic Programming (Lecture Notes in Computer Science)*, Malcolm I. Heywood, James McDermott, Mauro Castelli, Ernesto Costa, and Kevin Sim (Eds.). Springer International Publishing, Cham, 294–310. https://doi.org/10.1007/978-3-319-30668-1_19

[48] Elias Schede, Jasmin Brandt, Alexander Tornede, Marcel Wever, Viktor Bengs, Eyke Hüllermeier, and Kevin Tierney. 2022. A Survey of Methods for Automated Algorithm Configuration. *Journal of Artificial Intelligence Research* 75 (Oct. 2022), 425–487. https://doi.org/10.1613/jair.1.13676

[49] Shinichi Shirakawa and Tomoharu Nagao. 2009. Evolution of Search Algorithms Using Graph Structured Program Evolution. In *Genetic Programming (Lecture Notes in Computer Science)*, Leonardo Vanneschi, Steven Gustafson, Alberto Moraglio, Ivanoe De Falco, and Marc Ebner (Eds.). Springer, Berlin, Heidelberg, 109–120. https://doi.org/10.1007/978-3-642-01181-8_10

[50] Rainer Storn. 1996. On the Usage of Differential Evolution for Function Optimization. In *Proceedings of North American Fuzzy Information Processing*. 519–523. https://doi.org/10.1109/nafips.1996.534789

[51] Thomas Stützle and Manuel López-Ibáñez. 2019. Automated Design of Metaheuristic Algorithms. In *Handbook of Metaheuristics*, Michel Gendreau and Jean-Yves Potvin (Eds.). Springer International Publishing, Cham, 541–579.

[52] Sonja Surjanovic and Derek Bingham. 2013. Optimization Test Functions and Datasets. https://www.sfu.ca/~ssurjano/optimization.html.

[53] Andrew M. Sutton, Monte Lunacek, and L. Darrell Whitley. 2007. Differential Evolution and Non-Separability: Using Selective Pressure to Focus Search. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO '07)*. Association for Computing Machinery, New York, NY, USA, 1428–1435. https://doi.org/10.1145/1276958.1277221

[54] El-Ghazali Talbi. 2009. *Metaheuristics: From Design to Implementation.* Wiley, Hoboken, NJ.

[55] Jorge Tavares and Francisco B. Pereira. 2012. Automatic Design of Ant Algorithms with Grammatical Evolution. In *Genetic Programming (Lecture Notes in Computer Science)*, Alberto Moraglio, Sara Silva, Krzysztof Krawiec, Penousal Machado, and Carlos Cotta (Eds.). Springer, Berlin, Heidelberg, 206–217. https://doi.org/10.1007/978-3-642-29139-5_18

[56] Braden Tisdale, Deacon Seals, Aaron Scott Pope, and Daniel R. Tauritz. 2021. Directing Evolution: The Automated Design of Evolutionary Pathways Using Directed Graphs. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '21)*. Association for Computing Machinery, New York, NY, USA, 732–740. https://doi.org/10.1145/3449639.3459328

[57] Shengyin Wang and Kang Tai. 2004. Graph Representation for Structural Topology Optimization Using Genetic Algorithms. *Computers & Structures* 82, 20 (Aug. 2004), 1609–1622. https://doi.org/10.1016/j.compstruc.2004.05.005

[58] David H. Wolpert and William G. Macready. 1997. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation* 1, 1 (April 1997), 67–82. https://doi.org/10.1109/4235.585893

[59] Jonathan Wurth, Helena Stegherr, Michael Heider, Leopold Luley, and Jörg Hähner. 2023. Fast, Flexible, and Fearless: A Rust Framework for the Modular Construction of Metaheuristics. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation (GECCO '23 Companion)*. Association for Computing Machinery, New York, NY, USA, 1900–1909. https://doi.org/10.1145/3583133.3596335

[60] Wenjie Yi, Rong Qu, Licheng Jiao, and Ben Niu. 2023. Automated Design of Metaheuristics Using Reinforcement Learning Within a Novel General Search Framework. *IEEE Transactions on Evolutionary Computation* 27, 4 (Aug. 2023), 1072–1084. https://doi.org/10.1109/TEVC.2022.3197298

[61] Qi Zhao, Qiqi Duan, Bai Yan, Shi Cheng, and Yuhui Shi. 2024. Automated Design of Metaheuristic Algorithms: A Survey. https://doi.org/10.48550/arXiv.2303.06532 arXiv:2303.06532 [cs]

[62] Qi Zhao, Bai Yan, Xianglong Chen, Taiwei Hu, Shi Cheng, and Yuhui Shi. 2023. AutoOpt: A General Framework for Automatically Designing Metaheuristic Optimization Algorithms with Diverse Structures. https://doi.org/10.48550/arXiv.2204.00998 arXiv:2204.00998 [cs]