FLIMCS framework implementation guidelines

Nathan van der Westhuyzen

October 29, 2024

Contents

This report serves as an introductory guide to the implementation of the *fitness landscape-integrated metaheuristic configuration selection* (FLIMCS) framework. The report opens with an introduction in §1 providing the context for where the FLIMCS framework holds its value. The discourse continues in §2 with an overview of the framework with reference to a high-level data flow diagram. Next, the detailed implementation of the framework in R is presented in §3. Here, the online phase is presented first in §3.1, where the script and implementation is defined. Finally, the (so-called) offline phase is described in the scenario where an additional problem class is to be added.

1 Introduction

The number of metaheuristic solution methodologies available in the literature for solving combinatorial optimisation problems has increased significantly since the 1970s. This wide variety of approximate optimisation techniques include *tabu search*, *simulated annealing* (SA), *variable neighbourhood searches*, *genetic algorithms* (GAs), *ant colony optimisation*, and *particle swarm optimisation*, to name but a few. This leaves researchers and analysts with the challenging and time-consuming task of selecting an appropriate metaheuristic from the vast array of methods at their disposal when attempting to solve a particular optimisation problem (approximately). To exacerbate this challenge, it has been shown that there is typically no algorithm which dominates all other algorithms for all problem instances, but rather that different algorithms perform best in respect of

different types of problem instances. This is the phenomenon of *performance complementarity* — which has been observed for the majority of NP-hard optimisation problems — and should not be mistaken with the *no free lunch* theorems[1] of Wolpert and Macready.

As a result of the large variety of algorithmic alternatives at one's disposal, and the phenomenon of performance complementarity, a natural question arises as to whether automation of this challenging and intricate process is a viable option. The situation described above has given rise to the research field of *automated algorithm selection* (AAS) which has centred on the *algorithm selection problem* (ASP) originally proposed by Rice in 1976.

The problem of *algorithm selection* (AS) and the closely-related, yet distinct, problem of *algorithm configuration* (AC) are aimed at identifying and selecting the best competitor from among a set of algorithms (or from an *algorithm portfolio*) based on some measure of performance. An alternative approach is to consider a parallel algorithm portfolio — the situation in which a set of algorithms is executed in parallel upon which the algorithm is selected which yields the best result. This approach has shown promising results but is associated with the significant disadvantages of high computational burden and extensive implementation efforts required. These disadvantages highlight the need for a computationally efficient and accurate AS or AC approach tailored towards decision support for metaheuristic implementation.

Successful AS and AC models have more recently leveraged the powerful approach of machine learning to make connections between problem instance characteristics and algorithmic performance. This is possible through *meta-learning*, during which so-called *meta-features* are extracted from problem instances to represent the unique "characteristics" of the problem instances. Thereafter, the meta-features are used to train machine-learning models capable of predicting metaheuristic performance when solving unseen optimisation problem instances of a particular class. Based on these predictions, an algorithm or algorithmic configuration may then be selected that is expected to perform best in a particular context.

The research aim in this dissertation is to design, develop, implement, and evaluate the utility of a generic `FLIMCS` framework capable of facilitating simultaneous AS and AC decision support in the context *binary programming problems* (BPPs). The framework is applicable to a wide variety of BPP instances across various application classes and provides a clear roadmap for supplying analysts with repeated and dynamic decision support in aid of selecting appropriately configured metaheuristics for approximately solving unseen BPP instances. The framework utilises generic *fitness landscape analysis* (FLA) measures tailored to the general form of a BPP and mitigates the effects of performance complementarity by the adoption of a flexible algorithm portfolio with the option of including different algorithms while simultaneously considering different functionality options and hyperparameter configurations for these algorithms. Furthermore, AS and AC recommendations are provided by the framework based on a bi-objective consideration during which expected solution quality and anticipated computational burden are assessed in a trade-off fashion.

2 The `FLIMCS` framework

The `FLIMCS` framework is aimed at supplying analysts with repeated and dynamic decision support in aid of selecting appropriately configured metaheuristics for approximately solving unseen instances of some user-specified optimisation problem class in the binary programming domain. As mentioned, the challenge of selecting and tuning the hyperparameters of a metaheuristic for application to real-world optimisation problems is experienced by many, if not all, analysts in industry. It is therefore envisaged that the framework underpins full-cycle decision support from analysis to selection to hyperparameter tuning (*i.e.* full metaheuristic configuration). The `FLIMCS` framework architecture is illustrated graphically in Figure 1 in the form of a high-level DFD.

As indicated in the figure, the framework comprises two distinct phases, namely an *offline* phase and an *online* phase, which both interact with a meta-learning *database*. A framework user interacts with both phases as well as with the meta-learning database. The two phases function asynchronously and in alternating fashion, with

[1]The no free lunch theorems are based on the notion that since no two problem instances are identical, no single optimisation algorithm can reasonably be expected to perform the best (be superior) for all optimisation problem instances.

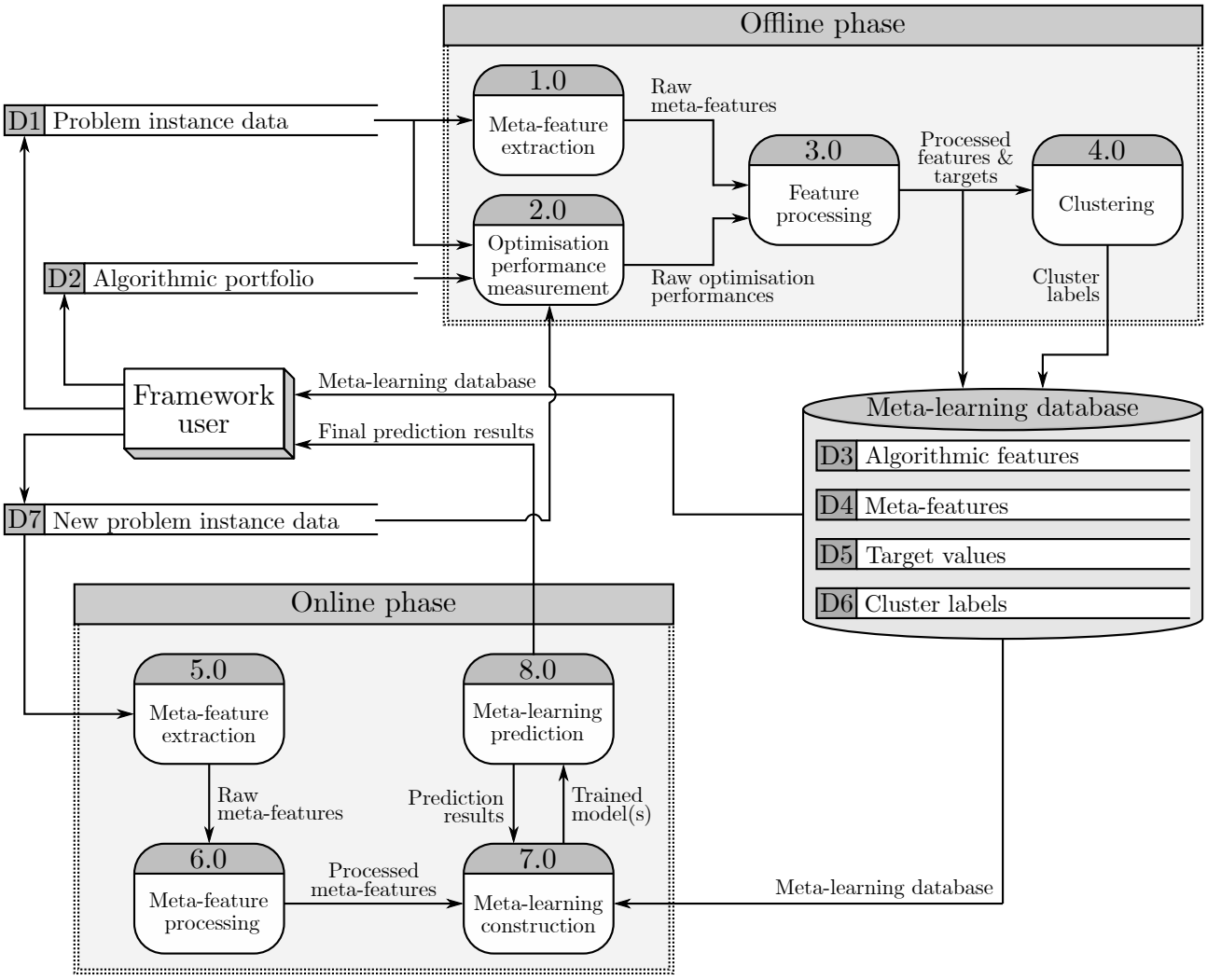

Figure 1: A high-level DFD of the proposed `FLIMCS` framework.

the offline phase initially preceding the online phase. The offline[2] phase prepares the database during the first iteration of framework execution for use during the subsequent online[3] phase. In later iterations, the database is updated during the offline phase.

As suggested by the name, the offline phase is responsible for capturing input data, performing FLA and other analyses, measuring metaheuristic optimisation performance in order to establish a detailed database of meta-features or to update an existing database of meta-features, and clustering input optimisation problem instances for more effective meta-learning model construction. More specifically, the offline phase takes as input a set of optimisation *problem instance data* within the binary programming domain and a user-specified *algorithmic portfolio* (stored in Data stores D1 and D2, respectively). The optimisation problem instances are analysed individually in terms of different fitness landscape measures so as to facilitate the extraction of raw meta-features, and then the various metaheuristic configurations in the algorithmic portfolio are executed with a view to establish optimisation performances in the context of the aforementioned optimisation problem instances. Next, the *raw* meta-features and optimisation performances, along with the algorithmic portfolio, are processed and transformed into *algorithmic features*, *meta-features* and *performance targets* (stored in Data stores D3, D4, and D5, respectively). Thereafter, the processed meta-features are utilised to *cluster* the input optimisation

[2] *Offline*, in this case, refers to the phase being performed with respect to a fixed, stationary data set in order to construct a meta-learning database, which is subsequently employed to build an AC and AS recommender.

[3] *Online*, in this case, refers to the phase being performed interactively for the purpose of generating metaheuristic decision support.

problem instances by assigning suitable cluster labels to them (stored in Data store D6). Finally, Data stores D3–D6 are combined so as to establish (or update) a detailed meta-learning *database*.

The offline phase forms the computational foundation of the framework, as it is responsible for generating a knowledge base combining all meta-features, algorithmic configurations in the portfolio, and cluster labels in respect of the input optimisation problem instances for performance measurement purposes. The database compiled during the offline phase is fundamental to the application of the subsequent online phase.

Building on the offline phase, the online phase takes two inputs: An unseen *input optimisation problem instance* from the binary programming domain (stored in Data store D7) and the entire *database* compiled or updated during the preceding offline phase. During the online phase, the same fitness landscape measures as those utilised during the offline phase are computed for the unseen optimisation problem instance. Thereafter, these raw features are transformed into meta-features in the same manner as during the offline phase. The processed meta-features thus obtained, along with the entire meta-learning database, are then taken as input for the construction of tailored meta-learning prediction models (trained and validated in respect of the database). This involves identifying the most suitable cluster of optimisation problem instances in the meta-learning database for the new problem instance, preparing training data, and training meta-learning prediction models. A single meta-learning model is crafted for each metaheuristic in the algorithmic portfolio. This approach is necessary, because each algorithm contains a collection of algorithmic features, which cannot be transferred between algorithms. Finally, once satisfactory levels of meta-leaning prediction model training accuracy have been achieved, the models are employed to generate predictions for the new problem instance which are returned to the framework user.

After all meta-learning predictions have been performed, the input problem instance is injected into the offline phase so as to obtain true optimisation performance measures. The database is then updated with the new set of performance measures and meta-features obtained during a re-application the offline phase. Lastly, the clustering procedure is repeated to identify new formations of clusters, as well as any inclusions of new data points to existing clusters identified during the previous offline phase execution. This is done in a bid to render the meta-learning database more and more representative over time. The updating phase of the database in terms of including new, unseen problem instances should ideally only be invoked in batches from time to time when the time-consuming task of collecting metaheuristic optimisation performances is deemed feasible.

Just as the offline phase is the computational foundation of the framework, the online phase may be thought of as a consulting house built upon that computational foundation. This consulting house provides a platform for user interaction, facilitating the extraction and processing of unseen meta-features, and provides as output AS and AC recommendations aimed at guiding users during metaheuristic optimisation implementations.

3 Framework implementation

This section is devoted to the detailed description of the `R` implementation available on GitHub[4]. The operational framework, in the context of four classical BPP classes — the *travelling salesperson problem* (TSP), the *knapsack problem* (KP), the *graph colouring problem* (GCP), and the *1-dimesional bin packing problem* (1DBPP). The problem data is available on the `MATILDA`[5] research platform. This platform makes problem instance data, problem-specific meta-data, and related publications available for a variety of problem classes. More information on the each of the problem classes and the methods of instance generation may be found in the publications cited in Table 1.

A set of 13 FLA measures, as well as an algorithmic portfolio comprising two well-known metaheuristics, are considered in this `FLIMCS` framework implementation. A summary of the meta-feature set, comprising the feature numbers and parameters values considered, may be found in Table 2. The complete algorithmic portfolio is summarised in Table 3. The portfolio contains 1 944 distinct GA configurations and 2 430 SA configurations, totalling to 4 374 distinct algorithmic configurations.

[4] `https://github.com/NathanvdWesthuyzen/FLIMCS`.
[5] `https://matilda.unimelb.edu.au/matilda/`.

Table 1: The sampling context of the optimisation problem classes considered, along with sources providing further detail on each problem class.

Problem class	Available instances	Sampled instances	Publication
TSP	1 330	950	Smith-Miles and Tan [5]
KP	5 300	1 060	Smith-Miles *et al.* [4]
GCP	8 278	1 000	Smith-Miles *et al.* [6]
1DBPP	8 815	1 000	Liu [3]

Table 2: A summary of the meta-feature set, including the feature number and the various relevant parameter values considered.

Number	Name	Parameters
1–3	HDIL	$\boldsymbol{L}_c \in \{L_1, L_2, L_3\}$
4	Constraint violation severity	None
5–14	Autocorrelation	$d \in \{1, \ldots, 10\}$
15	Correlation length	None
16–24	FEM	$\boldsymbol{\varepsilon} \in \{0, \frac{\varepsilon^*}{128}, \frac{\varepsilon^*}{64}, \frac{\varepsilon^*}{32}, \frac{\varepsilon^*}{16}, \frac{\varepsilon^*}{8}, \frac{\varepsilon^*}{4}, \frac{\varepsilon^*}{2}, \varepsilon^*\}$
25–33	SEM	
34	AEP	None
35–39	Hamming distance feasibility measure	$d_H = 1$ for $\mathcal{S}_s\|_{\text{any}}$
40–43	Pocket size measure	$p_{\max} = 100$ for $\mathcal{S}_s\|_{\text{feas}}$ and $\mathcal{S}_s\|_{\text{infeas}}$
44–45	Population evolvability metric	None
46–47	Population information content	None
48	Change rate	None
49–50	Negative slope coefficient	None

3.1 Online phase

The offline phase can be controlled through the script named Online_Phase.R, through which the online phase can be employed on the existing meta-learning database. Here, the user can select the following for experimentation:

target_variable defines which target variable is considered for prediction. For solution quality, set it to "norm_f", and for run time, set it to "norm_t."

k_size selects which *number of clusters* to consider. The larger the more computationally efficient, as smaller clusters are formed and induce a shorter computational run time during model training. The distribution of clusters can be observed in Figure 2, where larger clusters require excessive computing (and thus, cloud computing resources).

no_cores defines the number of parallel threads to employ (not to be set to more than the number available). A minimum of 32 threads is recommended for smooth implementation — a computing cluster is also recommended with sufficient memory resources.

verification_instance specifies which of the 120 verification instances are considered for experimentation.

The script opens with the possibility of user engagement (by altering the above parameters). Thereafter, the necessary information is initialised and imported (Lines 50–81). Next, the cluster which is best suited to the verification instances under consideration is identified (in Lines 83–99). Here, the clustering results (available in the Clustering_Results directory) are utilised to identify which cluster centroid is nearest to the new problem instance.

Table 3: A two-algorithm portfolio comprising SA and a GA. A two-minute kill switch was imposed on both metaheuristics for efficiency of optimisation performance collection.

Metaheuristic	Functionality	Options	Hyperparameters
SA	Search initialisation	Random	—
		Hybrid	—
	Move operator	Local	—
		Global	—
		Probabilistic	$\theta \in \{1.0, 0.5, 0.1\}$
	Temperature schedule	Geometric cooling	$\alpha \in \{0.85, 0.92, 0.99\}$
		Geometric reheating	$\beta' \in \{1.5, 1.2, 1.05\}$
	Epoch control	Solution acceptances	$c_{\max} \in \{5, 10, 30\}$
		Solution rejections	$d_{\max} = c_{\max} + d$, for $d \in \{1, 3, 5\}$
	Search termination	Hybrid	Static ($t_{\max}$)
			Dynamic ($e_p^{\max}$)
			Run time limit (2 min)
	Constraint handling	Rejection	—
		Penalisation	—
		Repair	—
GA	Population size		$\mu \in \{20, 40, 60\}$
	Initial population	Random	—
		Hybrid	—
	Selection strategy	Tournament	$k_T \in \{3, 10\}$
		Rank-based	$s_p \in \{1.5, 2\}$
	Move operator	Crossover with mutation	$p_c \in \{0.4, 0.7, 0.95\}$
			$p_m \in \{0.01, 0.05, 0.1\}$
	Replacement strategy	Steady-state	$\lambda = 1$
		Elitism	$\lambda \in \{2, 8\}$
	Search termination	Hybrid	Static ($t_{\max}$)
			Dynamic (t_{non})
			Run time limit (2 min)
	Constraint handling	Rejection	—
		Penalisation	—
		Repair	—

Next, the models are to be constructed (or imported from the Trained_Models directory). This is controlled in Lines 107–204. If the models (one for GA prediction, and another for SA prediction) already exist, they are then imported (Lines 110–113), else the models are constructed according to the regime defined below. Once constructed, the models are saved for (possible) future employment in Lines 200–201.

The meta-learning model employed was the popular *random forest* model [1] which has been demonstrated to be the most robust machine-learning model for algorithmic performance prediction [2]. More specifically, the `ranger` random forest model [7], a fast-implementation variant available through the `ranger` package in `R`, was used exclusively.

Model training and tuning was conducted according to a five-fold cross-validation approach. This involved randomly partitioning the data set into training and test sets, with 80% of the instances allocated to training and the remaining 20% to testing. The partitioning was repeated five times, each time taking a different 20% subset as the test set. As a result, each problem instance forms part of the test set exactly once across the five iterations. Alongside the cross-validation, a portfolio of hyperparameter configurations for the `ranger` model were evaluated in an attempt to tune the meta-learning model. The MAE metric was utilised exclusively during training, as it provided an accurate measure of prediction

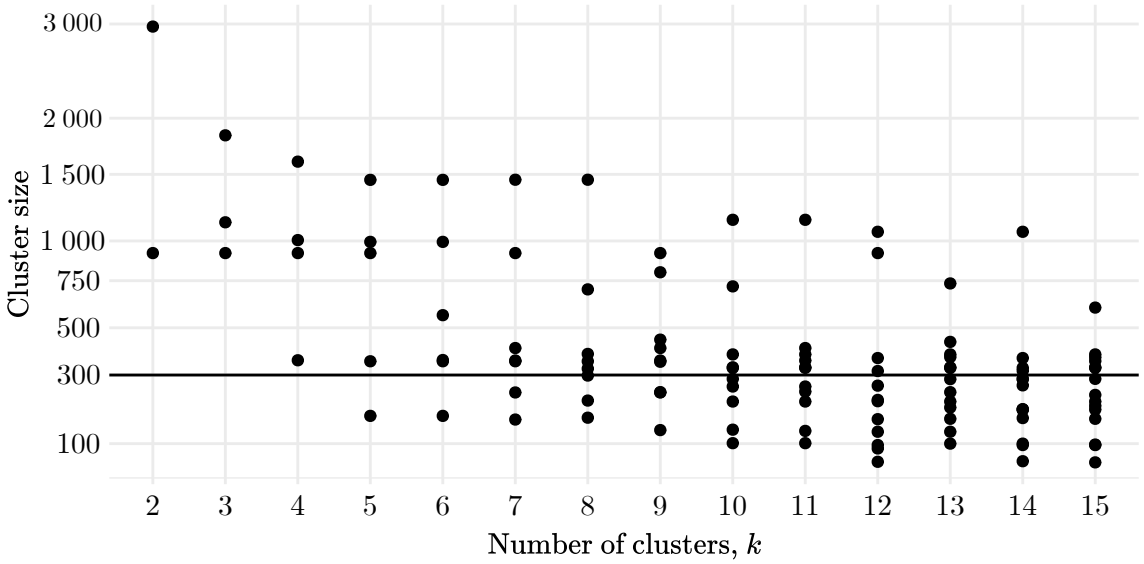

Figure 2: The cluster sizes for each number of clusters k. A local-execution threshold of three hundred is included for visual reference purposes.

error across the entire algorithmic portfolio. The MAE values obtained were then aggregated across each fold, thereby ensuring fair evaluation across different scenarios. The hyperparameter configuration that yielded the best aggregated performance was subsequently selected (as the trained and tuned model) for application to the verification problem instances. Each meta-learning model underwent individual training and hyperparameter tuning to ensure a suitable level of performance for each metaheuristic and target value.

The hyperparameter portfolio, outlined in Table 4, encompassed 54 distinct configurations of the `ranger` model. Four key hyperparameters were considered for tuning purposes, namely the *number of trees*, the *number of variables on which to split*, the *minimum node size*, and the *splitting rule*. Each of these parameters influences a specific aspect of constructing a random forest model. The number of trees hyperparameter, for instance, determines the number of individual trees to include in the forest, while the splitting rule governs how data are partitioned from a parent tree-node into child nodes. Three values were evaluated for each of the tree size, variable split number, and minimum node size hyperparameters. In addition, two splitting rules were examined: Maximally selected rank statistics (denoted by variance) and extremely randomized trees(denoted by extratrees). The default variable split number was taken as the rounded-down square root of the number of input variables, denoted by $\breve{x}$. Finally, the default minimum node size was taken as 5 for regression prediction, with one larger value and one smaller value also considered during tuning.

Table 4: The `ranger` hyperparameter configuration portfolio, where $\breve{x}$ denotes the rounded-down square root of the number of input features.

Hyperparameter	Model variable	Values
No. of trees	*num.trees*	$100, 200, 300$
No. of variable to split	*mtry*	$0.5\breve{x}, \breve{x}, 1.5\breve{x}$
Minimum node size	*min.node.size*	$3, 5, 7$
Splitting rule	*splitrule*	variance, extratrees

Finally, the trained models can be employed on the verification instance under consideration. Here, the models generate predictions which are considered in the context of both traditional performance (through metrics such as the *mean absolute error*), and recommendation performance (through tailored measures).

The first recommendation metric measures the true optimisation performance of the configuration variants that are estimated to perform best. The second recommendation metric is based on true algorithmic rankings and is aimed at verifying how close recommendations are to the best possible option. These novel metrics are only applicable to the solution quality target variable $\hat{f}$, while the traditional accuracies provide sufficient insight into the prediction of the run time target variable $\hat{t}$. Moreover, the solution quality predictions for both metaheuristics were again combined in order to evaluate the entire algorithmic portfolio fairly.

The novel accuracy metrics were employed in two ways: Evaluating the configuration variants that are expected to perform best (that is, the *top-performing* configuration variant), and those expected to perform within the top 1% of the algorithmic portfolio (that is, among the top 44 of the 4 374 configuration variants). These configuration variants represent the *recommendation* that a meta-learning model would make for metaheuristic implementation purposes. For all accuracy measures employed, a smaller value is indicative of superior meta-learning performance.

Finally, the script is closed by saving the prediction results in the Prediction_Results directory. This includes the aforementioned metrics along with a graphic of the prediction accuracy in a list format. The graphic illustrates the predictions relative to the *ground truth*, with the true values on the horizontal axis, and the predicted values on the vertical axis.

3.2 Adding a new problem class (offline phase)

The addition of another BPP class can be achieved through the adaption of four scripts, available under the New_Problem_Class directory. First, *meta-feature extraction* is achieved through the sequential execution of the RandomWalk_Generation.R and FLA.R scripts. Next, *optimisation performance measurement* is achieved through the execution of the GA.R and SA.R scripts.

Moreover, six *general function scripts* are provided, each encompassing a variety of crucial functions that ensure each component can operate in a modular and easy-to-maintain fashion. These functions are contained within the Functions directory. These function scripts need to be adapted in alignment with the new problem class. The necessary adpations are highlighted in each script.

3.2.1 Meta-feature extraction

The execution of the meta-feature extraction component is handled by first executing the RandomWalk_Generation.R script, where the necessary random walks are conducted through the problem instances considered. The new instances are considered by altering the INSTANCES parameter (indicating the ID linked to the importation of the problem data). Other crucial parameters include the following (to be set within both scripts):

no_cores defines the number of parallel threads to employ (not to be set to more than the number available). A minimum of 32 threads is recommended for smooth implementation — a computing cluster is recommended.

no_sols defines the number of random solutions to generate — that is, random, random feasible, and random infeasible solutions.

walk_length defines the number of steps to take for each random walk.

no_walks defines the number of random walks to execute (linked to the number of parallel threads available).

pop_walk_length defines the number of generations to evolve for each population-based walk.

moves defines the set of move operators considered, namely a *local* move, a *global* move, and three *probabilistic* moves that combines the two moves according to a reducing probability of employing the global move.

The FLA.R imports the necessary FLA techniques (functions) in Lines 74–92. These are available in the FLA_Techniques, and GA_Functions directories. An overarching for loop is imposed (from Lines 99–347) that

controls which problem instances are considered, looping through the set considered. Finally, the newly extracted FLA data are compiled, labelled, and saved (in RDS format) in Lines 322–341, under the FLA_Results directory.

3.2.2 Optimisation performance measurement

The execution of optimisation performance measurement component is handled in two scripts, one for each of the considered metaheuristics in the algorithmic portfolio. Both scripts operate in the same manner, selecting which new instances to collect optimisation performances for, and executing the algorithmic portfolio in parallel for each problem instance.

Both scripts open by requiring user inputs:

no_cores defines the number of parallel threads (cores) that are available.

INSTANCES selects which instances are considered (corresponding to the instances IDs).

timeLim specifies the run time limit (in seconds) imposed on each metaheuristic configuration variant.

The script opens with the importation of various algorithm-specific functions (available in the GA_Functions and SA_Functions directories). The algorithmic portfolios are defined next, with the termination criteria requiring user intervention, based on the computational burden of solving instances.

The output of each configuration includes a two-element list — each element corresponds to the optimisation result if the search process was terminated at the indicated point. This allows users to track the progress of the search relative to the defined termination criteria. The first element corresponds to *static* termination, while the second corresponds the *dynamic* termination. The run time is also returned.

3.2.3 Feature processing

The final component, feature processing, must be executed in alignment with the meta-learning databases. First, the FLA-based meta-features should be processed and combined with the algorithmic features (as in the existing databases). This process is conducted separately for the SA meta-data and the GA meta-data, since the two sets of algorithmic features are notably different and are not transferable between algorithms. Each problem instance and algorithmic configuration pair must then be matched with the optimisation performance measured.

Finally, the performances are to be normalised for each problem instance — min-max normalisation with a value of 0 representing the best performance and a value of 1 representing the worst performance obtained for that problem instance. For the computational run time target, a value of 0 corresponds to zero seconds of run time, while a value of 1 corresponds to the maximum run time of 120 seconds.

References

[1] L Breiman. "Random forests". In: *Machine Learning* 45 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324.

[2] F Hutter et al. "Algorithm runtime prediction: Methods & evaluation". In: *Artificial Intelligence* 206 (2014), pp. 79–111. DOI: 10.1016/j.artint.2013.10.003.

[3] K Liu. "Using instance space analysis to study the bin packing problem". MSc Thesis. University of Melbourne, Melbourne, 2020.

[4] KA Smith-Miles, J Christiansen, and MA Muñoz. *Revisiting where are the hard knapsack problems? via instance space analysis*. Computers and Operations Research, **128**, Manuscript 105184. 2021. DOI: 10.1016/j.cor.2020.105184.

[5] KA Smith-Miles and TT Tan. "Measuring algorithm footprints in instance space". In: *Congress on Evolutionary Computation*. Brisbane, 2012, pp. 1–8. DOI: 10.1109/CEC.2012.6252992.

[6] KA Smith-Miles et al. "Towards objective measures of algorithm performance across instance space". In: *Computers and Operations Research* 45 (2014), pp. 12–24. DOI: https://doi.org/10.1016/j.cor.2013.11.015.

[7] MN Wright and A Ziegler. "`ranger`: A fast implementation of random forests for high dimensional data in C++ and R". In: *Journal of Statistical Software* 77.1 (2017), pp. 1–17. DOI: 10.18637/jss.v077.i01.