EasyLocal++ a 25-year Perspective on Local Search Frameworks

The Evolution of a Tool for the Design of Local Search Algorithms

Sara Ceschia
Intelligent Optimization Laboratory
DPIA - Università degli Studi di Udine
Udine, Italy
sara.ceschia@uniud.it

Francesca Da Ros
Intelligent Optimization Laboratory
DMIF - Università degli Studi di Udine
Udine, Italy
francesca.daros@uniud.it

Luca Di Gaspero
Intelligent Optimization Laboratory
DPIA - Università degli Studi di Udine
Udine, Italy
luca.digaspero@uniud.it

Andrea Schaerf
Intelligent Optimization Laboratory
DPIA - Università degli Studi di Udine
Udine, Italy
andrea.schaerf@uniud.it

ABSTRACT

EasyLocal++ is a white-box C++ framework for designing local search algorithms. Over the years, it has been successfully used across various domains, such as timetabling, rostering, scheduling, and logistics, and has produced state-of-the-art results in benchmark datasets and competitions. Beyond research, EasyLocal++ has found practical use in real-world and industrial settings, demonstrating the flexibility and adaptability of the framework for different applications. In this paper, we position EasyLocal++ within the existing literature by comparing its capabilities with those of available alternative/similar tools. We then trace its history from its initial design 25 years ago to the current version. Furthermore, we describe its architecture, highlighting its design principles and functionalities. We also discuss the features developed to simplify the design of local search methods and enhance their performance. Lastly, we explore potential future perspectives and developments.

CCS CONCEPTS

• **Theory of computation** → **Optimization with randomized search heuristics**; • **Software and its engineering** → **Development frameworks and environments**; • **Applied computing** → *Operations research.*

KEYWORDS

Software tools, Algorithm engineering, Local search, Metaheuristics

ACM Reference Format:
Sara Ceschia, Francesca Da Ros, Luca Di Gaspero, and Andrea Schaerf. 2024. EasyLocal++ a 25-year Perspective on Local Search Frameworks: The Evolution of a Tool for the Design of Local Search Algorithms. In *Genetic and Evolutionary Computation Conference (GECCO '24 Companion), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3638530.3664140

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

ACM ISBN 979-8-4007-0495-6/24/07
https://doi.org/10.1145/3638530.3664140

1 INTRODUCTION

Metaheuristics (MHs) have emerged as the method of choice for tackling numerous NP-hard combinatorial optimization problems. Given this great success, over the years, several MHs have been devised, for instance Simulated Annealing (SA) [49], Tabu Search (TS) [43], Genetic Algorithm (GA) [46], Evolutionary Algorithm (EA) [27], Ant Colony Optimization [37], etc. However, the No Free Lunch Theorem [72] stipulates that no single algorithm is the best across all problems and instances. Consequently, MH researchers face the imperative of evaluating various algorithms to identify the most suitable technique for a given problem. To navigate this panoply of algorithms, it would be helpful to turn to frameworks and libraries, which enable, in one single project, to implement and assess multiple MHs [70]. Since the nineties, a sprout of tools has been proposed [55]. Among the earliest in this sense, our research group has developed EasyLocal++, a C++ white-box framework designed to support the development and evaluation of Local Search (LS) algorithms.

We have been using and updating EasyLocal++ since its creation in 1999. We have employed it to successfully solve several problems in different domains, including timetabling [4, 6, 11] and rostering [16, 18], scheduling [15, 19, 21], logistics [20], facility location problems [22], bioinformatics [29], and finance [28]. EasyLocal++ has also been used in solution methods that obtained state-of-the-art results in the benchmark datasets of many problems [5, 13, 17, 69, 73], including the ITC-2021 competition[1] in which it ranked second [60]. Using EasyLocal++ as a software tool extends beyond research, encompassing real-world and industrial applications. For instance, it has been employed in emergency facility location within the Italian Health Ministry project EasyNet[2] [26], in industrial scheduling within the context of steel production in collaboration with Danieli Automation [3], and in nurse rostering [12] in collaboration with Windex.[3] In addition, EasyLocal++ is used in various applications developed by the software house EasyStaff,[4] which was initially a spin-off of the University of Udine.

[1] See https://robinxval.ugent.be/ITC2021/
[2] See https://easy-net.info/.
[3] See https://www.windex.it.
[4] See https:/www.easystaff.it.

In this paper, we aim to describe the development of the framework over its 25 years, including a description of its most recent features. Thus, the remainder of this paper is structured as follows. Section 2 provides an overview on similar MH frameworks. Section 3 presents the evolution of EasyLocal++ since 1999. Section 4 describes the overall architecture of the project. Section 5 details the most relevant features, whereas Section 6 outlines the teaching purposes of the tool. Finally, Section 7 draws some conclusions and outlines future directions.

Software Availability. EasyLocal++ is available on `GitHub` at the public repository https://github.com/iolab-uniud/easylocal. The stable version is tagged with v.3.3, whereas the newest is in the development branch v.4. The code is distributed under MIT license.

2 RELATED SOFTWARE

In many optimization paradigms like Integer Linear Programming (ILP) and Constraint Programming (CP), numerous widely accepted tools exist, including ILOG CPLEX and CP Optimizer, Gurobi, OR-Tools, Gecode, and MiniZinc, among others. Such universally accepted tools have not been available for LS algorithms, or more in general MHs, where researchers still rely on customized coding solutions [63–65]. In fact, these implementations are often tailored to the particular problem being addressed, hindering the re-usability of the code and the replicability of the experiments. Despite this common practice, as early as three decades ago, there were already software packages available for MH implementations, such as those proposed by Andreatta et al. [2], Vaessens et al. [68]. The first introduced a conceptual framework for LS based on design patterns, whereas the latter suggested a template to organize LS algorithms.

In Table 1, we consider and summarize the main characteristics of a selection of MH software. The tools are primarily implemented in languages such as C++ and Java, although some attempts also exist in Scala, C#, and Python. Among the various packages, notable distinctions are made between libraries (e.g., GAlib [71], GAUL [1]) and frameworks (e.g., HotFrame [40] and ParadisEO [38]), with the former emphasizing code reuse (where user-defined code calls methods in the library) and the latter focusing on code-architecture reuse (where the framework code calls user-defined code). A few attempts are also in the form of API [41] and Domain Specific Language (DSL) [45]. Certain tools are tailored to address a specific group of MHs; for instance, Emili [54] is exclusively dedicated to LS methods, while PyMOO [8] focuses on GAs. In contrast, other packages (e.g., ParadisEO) support several optimization paradigms.

Another relevant distinction of the tools concerns the support of different multi-criteria cost function evaluations. Some of them are capable of supporting Multi-Objective Optimization (MOO) (in the Pareto sense) instead of just dealing with single objective optimization. Furthermore, some tools provide support in the automatic design of algorithms [42] (e.g., Emili) rather than treating MHs as monoliths. Eventually, while most of the packages have been updated in the last few years, a few ones have not been maintained for quite some time (e.g., HotFrame). Note that, in this case, we refer to the most recent date among the last commit in the repository and the last publication exclusively describing the tool. For further disquisitions, we forward the interested reader to some literature reviews on the topic [50, 55, 58].

3 HISTORY

In this section, we provide an overview of the development of EasyLocal++, highlighting key events (see Figure 1).

In 1999, Schaerf et al. started developing a general framework for LS which resulted in Local++ [61]. This initial concept, built upon a hierarchy of abstract template classes, was further refined, expanded, and developed by Di Gaspero and Schaerf [31, 32, 33], leading to the creation of EasyLocal++. Although resembling the present version to some extent, its neighborhood functionalities were initially limited, primarily focusing on basic neighborhood handling capabilities. In its second version, EasyLocal++ began managing multi-modal neighborhoods, incorporating up to three neighborhoods via chains of `if` control structures. The adoption of a structured multi-neighborhood approach within EasyLocal++ was facilitated by its application to specific problem domains, such as the traveling tournament problem [35] and a plethora of timetabling problems [34]. This structured approach was later revised by Di Gaspero and Urli (see for further details Urli [67]) through template metaprogramming using recursion, ultimately giving rise to EasyLocal++ v.3. In this version, several improvements were made and the code was generally refactored (e.g., management of several data structures with shared pointers instead of stack-allocated objects).

Between 2018 and 2022, significant revisions and simplifications were made to EasyLocal++. These modifications included the creation of a header-only version of the code (leading to EasyLocal++ v.3.1), a major overhaul of the control structure for the SA algorithm (resulting in EasyLocal++ v.3.2), and refactoring in the handling of outputs together with the integration of learning mechanisms in the selection of neighborhoods in SA [14] (culminating in EasyLocal++ v.3.3, that is the current stable version).

Recently, additional key features have been added to the framework, including the integration of multi-objective algorithms [25] and automated algorithm design (resulting in an initial implementation of EasyLocal++ v.4, i.e., the development version).

EasyLocal++ provides not only an easy way to encode LS solution methods but also capabilities for easy testing of code and analysis of neighborhoods. In this sense, several add-ons have been developed to address ancillary activities over the years. For instance, EasySyn [36], a software tool for the automatic synthesis of the source code for a set of local search algorithms, and EasyAnalyzer [30]. Unfortunately, these initial efforts were tightly coupled with the framework code, and therefore, their development was left behind in the framework's evolution. Nevertheless, some of those concepts (e.g., testers, see Section 5.4) were integrated directly into the core of EasyLocal++.

Along the evolution of EasyLocal++, several spin-off software have been developed, including porting of the main LS modules in Java (JEasyLocal [59]) and Julia (JuLeS [24]). Additionally, we have pursued experiments with other MHs, like GAs with EasyGA [7], and hybridization, like the intersection of CP and LS in Gelato [23].

Despite being completely decoupled from specific problem domains (see Section 4.1), the development of EasyLocal++, the creation of software spin-off, and the addition of new features have been driven by the requirements of implementing specific applications. In this regard, its development can be viewed as bottom-up rather than top-down: as challenges arose during the application of

Table 1: Comparison of MH software, distinguishing by programming language, software type, implemented algorithms, multi-objective capabilities, automated design features, and last update.

Tool	Language	Type	MH	MOO	Automated design	Last update
EasyLocal++	C++	Framework	LS	✓	✓	2024
ECJ [62]	Java	Framework	LS, EA	✓		2022
Emili [54]	C++	Framework	LS		✓	2022
Fonseca & Jesus [41]	Python	API	LS, EA			2023
GALib [53, 71]	C++	Library	GA			2024
GAUL [1]	C	Library	LS, GA			2010
Hexaly [45]	DSL (API in Python, C++, C#, Java)	Modeling language	LS			2024
HeuristicLab [44]	C#	Framework	LS, EA, GA	✓		2024
Hotframe [40]	C++	Framework	LS			2002
jMetal [39]	Java	Framework	EA	✓	✓	2024
OscaR [56]	Scala	Framework	CP-LS			2023
ParadisEO [38]	C++	Framework	LS, EA	✓	✓	2022
PyMOO [8]	Python	Framework	EA	✓		2024

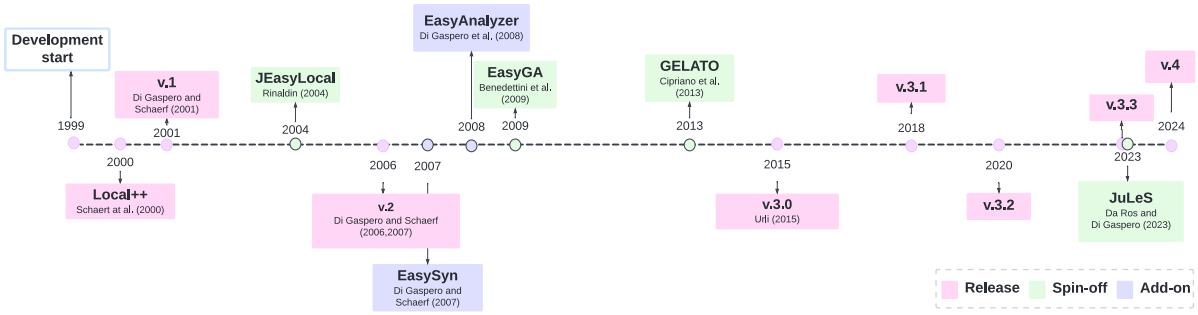

Figure 1: The evolution of EasyLocal++ since its first development in 1999.

EasyLocal++ to various optimization problems, we have continuously updated the framework by addressing missing functionalities and incorporating new elements and components, among other improvements. These efforts have made EasyLocal++ more robust, versatile, and suitable for a wide range of applications.

At present, EasyLocal++ is undergoing significant refactoring aimed at simplifying its codebase using modern C++ features (see Section 5.1). The ongoing development has brought EasyLocal++ to version 4, reflecting its continual evolution and adaptation to meet new requirements and needs. Nevertheless, the seminal design principles are kept untouched w.r.t. its first version.

4 ARCHITECTURE

EasyLocal++ is a white-box framework, transparent to the users and open-box by nature, specifically designed for handling LS algorithms. The internal logic and structure of the framework are fully exposed to and modifiable by the users. This transparency allows the user to understand, modify, and extend the functionalities of the framework with detailed insights into its workings.

In the following sections, we outline the design principles of EasyLocal++ (see Section 4.1). We then describe the algorithms that are available off-the-shelf (see Section 4.2) and discuss the concepts related to multi-neighborhoods (see Section 4.3).

4.1 Design Principles

EasyLocal++ primary design goals are, on the one hand, the easy prototyping of solution methods for the problem at hand (i.e., applying an off-the-shelf LS to the problem); on the other hand, the possibility to extend the framework without modifying the existing code (i.e., developing new MHs or new specific components).

The codebase is structured into distinct macro-modules, such as solvers and runners, each of which is further divided into modules (or classes). Each class is responsible for handling a specific aspect of LS (see Figure 2). This allows modularity, maintainability, scalability, and readability.

Modules within EasyLocal++ can be categorized based on their relevance to the specific optimization problem or their general applicability across different problems. The interface between these problem-specific and problem-independent modules relies on the *Inversion of Control* principle (also known as *Hollywood Principle*, recall Booch [9, p.143]): the control strategy of the MH is dealt with by the framework itself (see problem-independent part of Figure 2), while the user implements specific code related to the problem (see

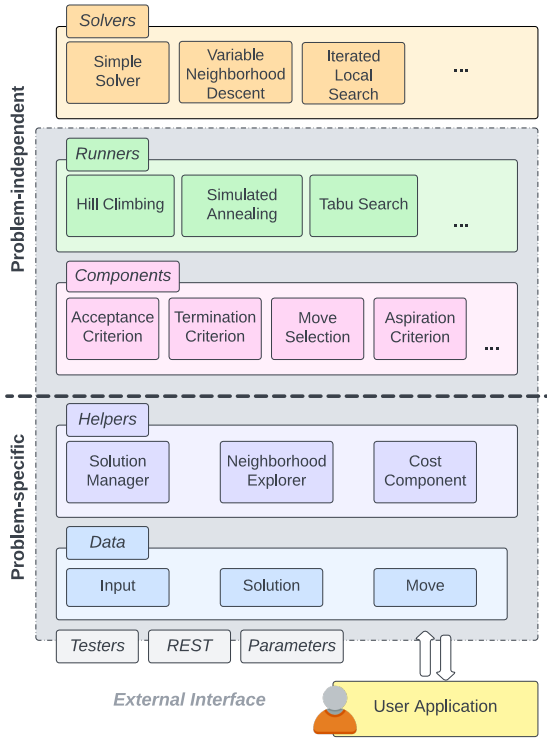

Figure 2: EasyLocal++ system architecture. A distinction between problem-specific and problem-agnostic parts of the framework is made.

problem-specific part of Figure 2). Eventually, the framework will call the user's code when necessary. This allows a loose coupling and a clear chain of responsibilities among the different classes and opens the door to the definition of abstract MH.

EasyLocal++ classes can also be organized considering the distinction provided by Pree [57]; therefore, accounting for *hot* and *frozen spots*. Hot spots represent areas where users are actively required to develop and customize functionalities based on problem-specific details (i.e., data and helpers in Figure 2). The frozen spots, conversely, are entirely implemented at the framework level and deal with problem-independent strategies (i.e., solvers, runners, and components in Figure 2). While the user cannot modify frozen spots, they can be easily extended (i.e., they are closed to modifications but open to extension [65]). For instance, the user can add a new component (e.g., a new termination criterion, a new acceptance criterion, etc.), develop a brand new LS method to be added to the runners or a different solving strategy (included in solvers).

In the following sections, we first present EasyLocal++ from the perspective of a user who aims to develop a LS solution method for a specific problem at hand (see Section 4.1.1) and then from the perspective of a user who wants to extend the framework (see Section 4.1.2).

4.1.1 Solving a Problem. We now present the system architecture from the perspective of a EasyLocal++ user who aims to develop a LS solution method for a specific problem at hand. For representative purposes, we consider a well-known benchmarking problem in combinatorial optimization, the Permutation Flowshop Scheduling Problem (PFSP).

The user has to start defining the *Data* of the problem at hand. The `Input` class takes care of the problem instance. In the case of the PFSP, reported in Listing 1, it will store the number of jobs, the number of machines, etc.

Listing 1: Implementation of the `Input` class for the PFSP.

```
class PFSP_Input {
public:
   // <...>
   size_t jobs, machines;
};
```

The solutions and their manipulations are explicitly represented through the concepts of `Solution` and `Move`. Conversely, other frameworks rely on an indirect representation of solutions and use standardized moves, which involve an encoding/decoding phase later (e.g., Biased Random-key Genetic Algorithm [66] relies on a $[0, 1]$-valued chromosome that needs to be decoded into an actual solution to the problem at hand).

For instance, a user tackling the PFSP with EasyLocal++ is required to implement a `Solution` class that deals with a vector to store the processing order of the jobs on the machines (see Listing 2), and a `Move` that describe the swap of two jobs in such a schedule (see Listing 3).

Listing 2: Implementation of the `Solution` class for the PFSP

```
class PFSP_Solution {
public:
   vector<size_t> schedule;
};
```

Listing 3: Implementation of the `Move` class for the PFSP, specifically the code regards the swap of two jobs in the schedule.

```
class PFSP_SwapMove {
public:
  int pos_i, pos_j;
};
```

Additionally, the user must define some classes specific to LS that must be embedded into the helpers macro-module. The generation of an initial solution should be implemented into a `SolutionManager` class, while the `NeighborhoodExplorer` class is in charge of the neighborhood exploration of a specific `Move`.

For instance, Listing 4 reports the co-routine method in the `NeighborhoodExplorer` related to the `PFSP_SwapMove` in charge of the complete enumeration of the neighborhood.

Listing 4: Implementation of the `neighborhood` generator for `PFSP_SwapMove` of the PFSP.

```
Generator<PFSP_SwapMove>
    PSFP_SwapNeighborhoodExplorer::Neighborhood(
    shared_ptr<const PFSP_Solution> sol) const {
  for (size_t pos1 = 0; pos1 < st->Jobs() - 1; ++
    pos1)
    for (size_t pos2 = pos1 + 1; pos2 < st->Jobs()
    ; ++pos2)
```

```
    co_yield {pos1, pos2};
}
```

The composition of multiple moves (e.g., a swap move used together with an insert move) is synthesized from the framework itself (see Section 4.3).

We assume a minimization problem, thus, as a consequence, the objective function consists of `CostComponents` that take care of the computation of the penalties for the problem. How the costs are evaluated depends on the objective function strategy (see Section 5.6); currently, the costs can be aggregated into a scalar value or compared with a lexicographic ordering or in the Pareto sense.

Since the methods provided by all these classes are crucial in terms of performance, in EasyLocal++ v.4, they are implemented through static polymorphism (see Section 5.1); that is, these elements are further supplied to problem-independent classes through template instantiation based on C++17 *concepts*. Indeed, when the user defines all the previously described elements, the implementation of an actual LS solution method simply consists of an instantiation of the suitable runner which takes care of the LS technique (e.g., TS, SA, Hill Climbing (HC)) and, optionally, of the solver which combines single runners in a more sophisticated solution strategy. Each runners implements a slight variation, based on the characteristics of the LS algorithm itself, of the abstract LS algorithm (depicted in Algorithm 1). The behavior of a given runner can be further specified through a palette of components. For instance, a variety of stopping criteria could instantiate the `Terminate` function, based on the user's needs. From the command line, the user can specify which version of a particular component to use, as demonstrated in the example shown in Listing 5. This capability facilitates the process of automatic algorithm design, as discussed in Section 5.3.

Listing 5: An example of a configuration command line for a EasyLocal++ implemented TS to the PFSP

```
./pfsp --seed 45 --instance DD_Ta035.txt --
    termination IdleIterationsTermination --
    aspiration-criteria AspirationByObjective --
    stop-exploration
    StopExplorationBestImprovement --generator
    FullNeighborhoodGenerator --timeout 1 --max-
    idle-iteration 100 --tabu-list RangeTabuList
    --min-iteration-tl 2 --max-iteration-tl 9
```

4.1.2 Extending the Framework. We now present the perspective of a user who aims to extend the framework. Such a user can tackle the frozen spots, thus they can add a new component that can be used by an existing runner, a new runner (therefore, a new LS algorithm), or a new solver (hence, provide a way of combining different LS algorithms).

The abstract code shown in Algorithm 1 depicts the typical procedure for LS – note that the colors highlighting different methods in the pseudo-code refer to the system architecture (see Figure 2). The abstract code relies on the specification of three primary generic data types (`Input`, `Solution`, and `CostFunction`). According to the LS scheme, firstly an `InitialSolution` is computed; afterward, the algorithm enters a loop in which a move is first selected (`SelectMove`), its contribution to the cost is computed (`ComputeΔCost`), and if it is suitable (`AcceptableMove`) it is performed. The procedure iterates until a termination criterion is satisfied (`Terminate`). This overall procedure is a skeleton that can be adapted to several different MH. Indeed, different LS metaheuristics, such as HC, SA, or TS, provide distinct implementations of those functions according to their specific exploration strategy, also adding new potential ones (e.g., in TS a function for dealing with the aspiration criteria is needed, while this is not necessary for HC and SA).

For instance, let us consider the case of adding to a runner for the HC algorithm. The C++ code is reported in Listing 6 and replicates almost *verbatim* the pseudo-code in Algorithm 1. In such a case, the new runner can be used coupled with the plethora of already encoded components or may want to add a new component to be called in an existing runner. As an example, Listing 7 shows the implementation of a termination criterion based on the number of idle iterations. Notice that this class will be subject to static polymorphism; therefore, in the LS code, it will be checked for the proper *concept* compliance.

Algorithm 1: Abstract LS procedure.

Hotspots: A specification of the `Input` I, the `Solution` S, the definition of a `Move` M, and a `CostFunction` F

function `LocalSearch`$\langle I, S, M, F\rangle$(inst: I):

- s_0 := `InitialSolution`$\langle I, S\rangle$(inst)
- $c_0 := F(\text{inst}, s_0)$
- $(s^*, c^*) := (s_0, c_0)$
- $i := 0$
- **while** ¬`Terminate`$\langle S\rangle(s_i, i)$ **do**
  - m := `SelectMove`$\langle S, M\rangle$(inst, s_i)
  - ΔF := `ComputeΔCost`$\langle S, M\rangle$(inst, s_i, m, F)
  - **if** `AcceptableMove`$\langle S, M\rangle(m, s_i, \Delta F)$ **then**
    - $(s_{i+1}, c_{i+1}) := (s_i \oplus m, c_i + \Delta F)$
    - **if** $c_{i+1} < c^*$ **then**
      - $(s^*, c^*) := (s_{i+1}, c_{i+1})$
  - $i := i + 1$
- **return** s^*, c^*

Listing 6: Implementation of a generic HC scheme.

```
void HillClimbing::Go(shared_ptr<const Input> in)
    {
  // current_solution_v is a lazy cost structure
    bridging the solution and its cost components
  current_solution_v = sm->SolutionValuePtr(sm->
    InitialSolution(in));

  while (!termination.terminate(this)) {
      // analogously current_move_v stores the
    solution, the move, and the Δcosts
      current_move_v = nhe->MoveValuePtr(
    select_move.select(this));
    if (accept_move.accept(this)) {
      // apply the move and store the new solution
      *current_solution_v = *current_move_v;
      idle_it = 0;
    } else
      idle_it++;
    it++;
  }
  // copy to the final solution
```

```
  final_solution_v = make_shared<SolutionValue>(*
    current_solution_v);
}
```

Listing 7: Implementation of a termination criterion based on idle iterations.

```
template <RunnerIdleIterT Runner>
class IdleIterationsTermination : public
    Parametrized {
public:
  // <...>
  bool terminate(Runner* r) {
    return r->idle_it > max_idle_it;
  }
protected:
  size_t max_idle_it;
};
```

4.2 Off-the-shelf Local Search Algorithms

While extending the framework with the implementation of new MHs is relatively straightforward (see Section 4.1.2), EasyLocal++ is already equipped with a portfolio of LS, related variants, and components. They range from simple forms of HC strategies, like steepest descent, first descent, and random descent, to more complex MHs. For example, it includes different versions of Late Acceptance Hill Climbing (LAHC) [10], Iterated Local Search [52], and TS [43]. As an example of components, one can equip a TS choosing among different tabu list implementations (e.g., variable length tabu list, frequency-based tabu status).

The MH supported with the largest number of different versions is SA, given that it has been employed in most of our research work. It has provided state-of-the-art results for many optimization problems. In detail, besides the classic version of SA [48], we introduced the *cut-off* mechanism that speeds up the early stages of the search and a reheating mechanism that restarts the annealing procedure. In addition, we propose several termination criteria in the form of components, which are based on the number of iterations, the final temperature, and the timeout.

All the available MHs are entirely implemented at the framework level. To be used, they require only the definition of a single object of the suitable type and suitable template instantiations.

4.3 Multi-neighborhood

One of the most remarkable features of EasyLocal++ is the possibility of synthesizing composite neighborhoods starting from the implementation of basic ones. This feature allows the development of multi-neighborhood search methods, which have proven very successful in various applications.

In detail, given two or more `NeighborhoodExplorer` classes, EasyLocal++ generates automatically, at compile time, the neighborhood obtained by their composition. The simplest, though the most effective, form of composition is the union of the neighborhoods, although in EasyLocal++ v.3.3 also, other forms, such as cartesian product and move chains, are possible. For example, when a union of different neighborhoods is defined, the best move is selected by enumerating all moves of all basic neighborhoods. In the case of a LS approach based on random moves, the current move is obtained by selecting, in turn, the basic neighborhood and the specific move inside that neighborhood. In this case, the selection of the basic neighborhood is not necessarily made with a uniform probability. On the contrary, it is often more effective to assign different probabilities and bias the search toward specific directions, but without giving away the possibility of making different move types occasionally. This mechanism is advantageous when some neighborhoods are computationally more expensive than others or when some neighborhoods are more suitable for diversifying the search. In contrast, others improve the value of the cost function. As for any other algorithm parameter, these probabilities can be exposed as parameters and thus tuned.

Finally, EasyLocal++ includes the option of adjusting online the rates of the selection of the different neighborhoods so that they do not need to be tuned in advance and may adapt to the different stages of the search process. Specifically, the probabilities of each neighborhood are updated using the recency-weighted average bandit algorithm, with a reward function that involves both the relative improvement in the objective function and the computational cost of neighborhoods. Recently, this feature has been exploited by Ceschia et al. [14] to obtain state-of-the-art results on two different timetabling problems.

5 FEATURES

This section provides an overview of the features and characteristics of EasyLocal++, highlighting their evolution over the years.

5.1 Evolution toward C++23 Compatibility

EasyLocal++ has been developed in C++, following the language evolution over the years, and its last updates (e.g., usage of coroutines in the neighborhood generation) use C++23 characteristics.

C++ offers several advantages for MHs. As an object-oriented language, C++ naturally promotes modularity, reusability, and maintainability, all essential characteristics for MH implementations [64]. Additionally, it features powerful template metaprogramming capabilities, enabling compile-time code generation and optimization, which helps implement generic algorithms and data structures with high performance and flexibility (e.g., in multi-neighborhood automatic synthesis).

Moreover, as a compiled language, C++ is known for its speed, which allows performing intensive computations efficiently. This is advantageous for LS algorithms, which often require numerous iterations and exploration of possibly large neighborhoods.

Lastly, C++ benefits from quite a large community of users; consequently, developers can rely on a wealth of online resources, potentially providing valuable support for their projects.

As mentioned, EasyLocal++ has evolved alongside the advancements in the C++ language. As for its latest version (i.e., EasyLocal++ v.4), the implementation now utilizes static polymorphism instead of classical virtual function polymorphism for hot spots, which is now fully supported in newer C++ versions. This shift was motivated by the availability of C++17 *concepts*, which allows for checking complex constraints in template instantiation.

The codebase has also been re-engineered to embrace functional-style programming, improving code organization and readability. Notably, neighborhood exploration now employs C++23 *coroutines*,

enabling a streamlined approach to move generation (i.e., all the code for move generation is now contained in a single function, whereas before, a complex iterator structure was necessary). Furthermore, EasyLocal++ v.4 implements lazy evaluation for cost function computation, ensuring that cost components will be computed only when needed and cost values are cached to prevent unnecessary recomputation. For instance, in the context of MOO within the Pareto front, if no other solution dominates a specific cost component of a given solution, there is no immediate need to compute the values of the other components for that solution.

5.2 Benchmarking

Using a framework like EasyLocal++ standardizes the underlying code and data structures used to describe problems, enabling the comparison of the performance of various MHs on a uniform basis. This standardization mitigates potential confounding factors from different implementations, thus ensuring a more accurate and fair performance comparison, as advocated by Swan et al. [64].

Specifically, EasyLocal++ promotes the comparability in two ways. On the one hand, the user can experiment with different algorithms tailored to the problem at hand without re-implementing problem-specific modules. This flexibility allows for efficient exploration of algorithmic variations while maintaining consistency in problem representation. On the other hand, it allows testing the same algorithm across diverse problem sets, ensuring that the underlying LS procedure remains consistent throughout. This capability enhances confidence in the reliability and generalizability of algorithmic results across different problem domains.

5.3 Automated Algorithm Design

One source of inspiration for EasyLocal++ is the Programming by Optimization (PbO) manifesto [47]. Such a paradigm advocates for a shift of perspective in software development, where the design of components is approached through optimization. This involves a systematic exploration of design alternatives toward the selection of an optimal configuration of the different components through the use of automated tools (e.g., `irace` [51]). PbO is organized into five levels (numbered from 0 to 4). The first two levels deal with existing software and expose just parameters and constants (a.k.a. magic numbers). The upper levels (from 2 to 4) advocate a tighter integration of PbO into software design.

Until EasyLocal++ v.3.3, the only design choices exposed were algorithmic parameters (e.g., in TS the length of the tabu list and the idle iterations in the termination criteria) in compliance with levels 0 and 1 of PbO. Starting with EasyLocal++ v.4, following Franzin and Stützle [42], we do not consider LS algorithms as monoliths anymore. Instead, the users can choose among different options for their design choices (i.e., components layer in Figure 2), which are exposed for the automated design tool. These design choices are resolved at compile time, meaning no computational overhead is added at running time. Therefore, a panoply of algorithms with a meaningful combination of the design alternatives can be generated, and the relevant one can be directly selected through the Command Line Interface (CLI). For instance, when using HC, the user can choose which type of termination criterion to use (e.g., *number of idle iterations*, *maximum number of iterations*, *timeout*).

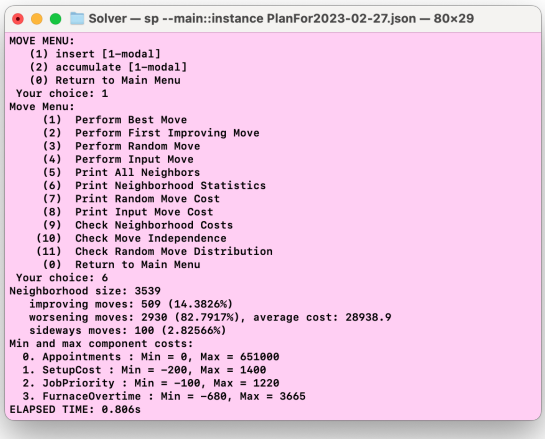

Figure 3: A screenshot of the text-based user interface provided by testers.

5.4 Testers

In addition to the CLI, the testers enable the automatic creation of a *text-based* user interface designed to facilitate multi-level user interaction with the software. This interface increases the software usability by supporting the debugging of problem-specific code, such as verifying the consistency of incremental evaluations within cost components. Moreover, it allows for preliminary analyses of helpers, including neighborhood statistics (Figure 3).

Also, the interface enables direct experimentation with various solution methods in a sequential manner.

5.5 Optimization as a Service

An alternative mode of interacting with solution methods, which arose while collaborating with industrial partners, involves using a REST API. In particular, EasyLocal++ solvers and helpers are encapsulated within an automatically generated REST HTTP interface, which enables their deployment as network-accessible services. Interactions with the API endpoints occur through the exchange of JSON-formatted files.

For instance, Listing 8 displays the main REST endpoint that provides information about the current state of a running system. Specifically, it shows the availability of a total of three solution methods (i.e., HC, TS, and SA), two different types of neighborhoods (i.e., accumulate and insert), and the execution status of a recently executed optimization task (i.e., the task, with the given identifier, regards the execution of a HC and is finished).

Listing 8: An example of main REST endpoint.

```
{ "runners": ["/runner/HC", "/runner/TS",
    "/runner/SA"],
  "neighborhoods": ["/move/accumulate",
    "/move/insert"],
  "tasks": [{"finished": true, "run_id": 55286,
    "runner": "HC"}] }
```

Listing 9 shows the outcome of the optimization task, which can be inspected through an additional REST request.

Listing 9: An example of task endpoint.

```
{ "cost": {
        "components": {
            "Appointments": 0,
            "FurnaceOvertime": 0,
            "JobPriority": 28,
            "SetupCost": 7
        },
        "objective": 35
    },
    "finished": true,
    "run_id": 55286, "runner": "HC" }
```

This approach shifts the focus from distributing the code to providing a quickly deployable optimization service and opens the possibility of building web applications for the interaction with the solution methods.

5.6 Multi-criteria Cost Function Evaluation

EasyLocal++ supports multi-criteria cost function evaluation in various forms, including aggregation (i.e., single-objective), hierarchical (i.e., lexicographic), and Pareto optimization. The framework accommodates single-cost components, which can be provided and integrated according to the desired optimization strategy.

Additionally, the implementation of lazy cost evaluation ensures that costs are computed only when necessary. The code in Listing 10 shows an example of this behavior: the access operator to a cost component of a `SolutionValue` element computes its cost only when it is accessed and, eventually, caches its value.

Listing 10: Lazy cost computation

```
T SolutionValue::operator[](size_t i) const {
  auto& val = const_cast<pair<bool, T>&>(this->at(
    i));
  if (!val.first) {
    val.second = cs->ComputeCost(sol, i);
    val.first = true;
  }
  return val.second;
}
```

6 EASYLOCAL++ FOR TEACHING

Since 2010, EasyLocal++ has played a significant role in teaching optimization within the *Advanced Scheduling Systems* course for the Master's Degree Program of Management Engineering at the University of Udine.[5] Over 13 years, around 200 students have received instruction on LS algorithms thanks to EasyLocal++. Students engage with the practical design and implementation of solution methods to real-world scheduling, timetabling, and routing problems. They are also encouraged to create their exam projects using EasyLocal++ to tackle specific optimization problems. The structured modularity and clear conceptual framework of EasyLocal++ have proven beneficial, particularly for students without a strong Computer Science background. These students have successfully implemented various techniques with reasonable programming efforts, allowing them to test and compare solutions across available instances.

7 CONCLUSIONS AND FUTURE WORK

We have described EasyLocal++, the white-box C++ framework for LS algorithms that our research group has developed since 1999.

Through its evolution to version 4, EasyLocal++ has consistently kept its foundational principles, including transparency to the user, modularity, maintainability, and readability, together with the separation of problem-specific and problem-independent classes. These principles have remained unchanged over its 25-year history.

The evolution, the addition, and the removal of features in EasyLocal++ have been primarily driven by its application to specific problems. Such an iterative development approach has ensured that the framework remains relevant, adaptable, and effective in addressing a wide range of optimization challenges across various application domains and w.r.t. different research trends. Indeed, over the years, it has proved effective in developing solution methods for real-world applications and academic benchmark problems belonging to a wide range of domains. In particular, it currently holds state-of-the-art results for examination timetabling, frequency assignment, home healthcare routing and scheduling, and medical student scheduling problems. In addition, thanks to its modular and clear system architecture, it has been successfully used for teaching purposes in the MSc of Management Engineering since 2010.

Looking to the future, several ways exist to enhance the capabilities of EasyLocal++. Firstly, the research presented by Ceschia et al. [14] can be expanded to incorporate other parameters of SA toward the concept of parameter-less algorithms. Secondly, performances could benefit from parallel executions. Additionally, there is potential to explore extensions towards other paradigms akin to LS, such as Large Neighborhood Search, which currently lack a unified software framework. Moreover, introducing hybrid approaches in the framework could offer promising avenues.

On a concluding note, while our project started 25 years ago, the interest in such software tools is far from outdated, as testified by the recent EU COST Action on Randomised Optimization Algorithms (with a specific reference to Working Group 1 – Problem modeling and user experience).[6] Indeed, one of the network's goals is to propose a transparent white-box modeling framework (and the related software implementations) for a broad range of real-world applications, bridging the gap between modeling and solving phases. These are the same goals that initially motivated the development of EasyLocal++.

ACKNOWLEDGMENTS

Over the years, EasyLocal++ has been developed and used by many people. We express our gratitude to those who have contributed to the project, including Stefano Benedettini, Raffaele Cipriano, Agostino Dovier, Paolo Rinaldin, Andrea Roli, Roberto Maria Rosati, Tommaso Urli, and Eugenia Zanazzo.

The work is carried out within the Artificial Intelligence 2022 – 2025 interdepartmental strategic plan project of the University of Udine.

[5] Videos of lectures covering LS and EasyLocal++ are available upon request.

[6] See https://roar-net.eu/.

REFERENCES

[1] Stewart Adcock. 2005. Genetic Algorithms Utility Library (GAUL). (2005). https://gaul.sourceforge.net/ [Accessed: 2024-04-05].

[2] Alexandre A. Andreatta, Sérgio E.R. Carvalho, and Celso C. Ribeiro. 1998. An object-oriented framework for local search heuristics. In *Proceedings. Technology of Object-Oriented Languages. TOOLS 26 (Cat. No.98EX176)*. IEEE, Santa Barbara, CA, USA, 33–45.

[3] Davide Armellini, Paolo Borzone, Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. 2020. Modeling and solving the steelmaking and casting scheduling problem. *International Transactions in Operational Research* 27, 1 (2020), 57–90.

[4] Michele Battistutta, Andrea Schaerf, and Tommaso Urli. 2017. Feature-based tuning of single-stage simulated annealing for examination timetabling. *Annals of Operations Research* 252 (2017), 239–254.

[5] Ruggero Bellio, Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. 2021. Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling. *Computers & Operations Research* 132 (2021), 105300.

[6] Ruggero Bellio, Sara Ceschia, Luca Di Gaspero, Andrea Schaerf, and Tommaso Urli. 2016. Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers & Operations Research* 65 (2016), 83–92.

[7] Stefano Benedettini, Andrea Roli, and Luca Di Gaspero. 2009. EasyGenetic: A Template Metaprogramming Framework for Genetic Master-Slave Algorithms. In *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, Second International Workshop, SLS 2009, Brussels, Belgium, September 3-4, 2009. Proceedings*, Thomas Stützle, Mauro Birattari, and Holger H. Hoos (Eds.). Lecture Notes in Computer Science, Vol. 5752. Springer-Verlag, Berlin-Heidelberg, Germany, 135–139.

[8] Julian Blank and Kalyanmoy Deb. 2020. pymoo: Multi-Objective Optimization in Python. *IEEE Access* 8 (2020), 89497–89509.

[9] Grady Booch. 1993. *Object-Oriented Analysis and Design with Applications (2nd Ed.).* Benjamin-Cummings Publishing Co., Inc., USA.

[10] Edmund K Burke and Yuri Bykov. 2017. The late acceptance hill-climbing heuristic. *European Journal of Operational Research* 258, 1 (2017), 70–78.

[11] Mats Carlsson, Sara Ceschia, Luca Di Gaspero, Rasmus Mikkelsen, Andrea Schaerf, and Thomas Stidsen. 2023. Exact and metaheuristic methods for a real-world examination timetabling problem. *Journal of Scheduling* 26 (2023), 353–367.

[12] Sara Ceschia, Luca Di Gaspero, Vincenzo Mazzaracchio, Giuseppe Policante, and Andrea Schaerf. 2023. Solving a real-world nurse rostering problem by Simulated Annealing. *Operations Research for Health Care* 36 (2023), 100379.

[13] Sara Ceschia, Luca Di Gaspero, Roberto Maria Rosati, and Andrea Schaerf. 2021. Multi-Neighborhood Simulated Annealing for the Minimum Interference Frequency Assignment Problem. *EURO Journal on Computational Optimization* 10 (2021), 1–32.

[14] Sara Ceschia, Luca Di Gaspero, Roberto Maria Rosati, and Andrea Schaerf. 2024. Reinforcement Learning for Multi-Neighborhood Local Search in Combinatorial Optimization. In *Machine Learning, Optimization, and Data Science*, Giuseppe Nicosia, Varun Ojha, Emanuele La Malfa, Gabriele La Malfa, Panos M. Pardalos, and Renato Umeton (Eds.). Springer Nature Switzerland, Cham, 206–221.

[15] Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. 2017. Solving Discrete Lot-Sizing and Scheduling by Simulated Annealing and Mixed Integer Programming. *Computers & Industrial Engineering* 114 (2017), 235–243.

[16] Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. 2023. Simulated Annealing for the Home Healthcare Routing and Scheduling Problem. In *AIxIA 2022 – Advances in Artificial Intelligence: XXIst International Conference of the Italian Association for Artificial Intelligence, AIxIA 2022, Udine, Italy, November 28 – December 2, 2022, Proceedings* (Udine, Italy). Springer-Verlag, Berlin, Heidelberg, 402–412.

[17] Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. 2023. Simulated Annealing for the Home Healthcare Routing and Scheduling Problem. In *AIxIA 2022 – Advances in Artificial Intelligence*, Agostino Dovier, Angelo Montanari, and Andrea Orlandini (Eds.). Springer International Publishing, Cham, 402–412.

[18] Sara Ceschia, Rosita Guido, and Andrea Schaerf. 2020. Solving the static INRC-II nurse rostering problem by simulated annealing based on large neighborhoods. *Annals of Operations Research* 288 (2020), 95–113.

[19] Sara Ceschia, Kevin Roitero, Gianluca Demartini, Stefano Mizzaro, Luca Di Gaspero, and Andrea Schaerf. 2022. Task design in complex crowdsourcing experiments: Item assignment optimization. *Computers & Operations Research* 148 (2022), 105995.

[20] Sara Ceschia and Andrea Schaerf. 2013. Local search for a multi-drop multi-container loading problem. *Journal of Heuristics* 19, 2 (2013), 275–294.

[21] Sara Ceschia and Andrea Schaerf. 2016. Dynamic patient admission scheduling with operating room constraints, flexible horizons, and patient delays. *Journal of Scheduling* 19, 4 (2016), 377–389.

[22] Sara Ceschia and Andrea Schaerf. 2024. Multi-neighborhood simulated annealing for the capacitated facility location problem with customer incompatibilities. *Computers & Industrial Engineering* 188 (2024), 109858.

[23] Raffaele Cipriano, Luca Di Gaspero, and Agostino Dovier. 2013. *A Multi-paradigm Tool for Large Neighborhood Search.* Springer Berlin Heidelberg, Berlin, Heidelberg, 389–414.

[24] Francesca Da Ros and Luca Di Gaspero. 2023. Exploring the Potential of JuLeS: A White Box Framework for Local Search Metaheuristics. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation* (Lisbon, Portugal) *(GECCO '23 Companion)*. Association for Computing Machinery, New York, NY, USA, 191–194.

[25] Francesca Da Ros and Luca Di Gaspero. 2023. Local Search Strategies for Multi-Objective Flowshop Scheduling: Introducing Pareto Late Acceptance Hill Climbing. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation* (Lisbon, Portugal) *(GECCO '23 Companion)*. Association for Computing Machinery, New York, NY, USA, 61–62.

[26] Francesca Da Ros, Luca Di Gaspero, Kevin Roitero, David La Barbera, Stefano Mizzaro, Vincenzo Della Mea, Francesca Valent, and Laura Deroma. 2024. Supporting Fair and Efficient Emergency Medical Services in a Large Heterogeneous Region. *Journal of Healthcare Informatics Research* (2024), 1–38. https://doi.org/10.1007/s41666-023-00154-1

[27] Kalyanmoy Deb, Ashish Anand, and Dhiraj Joshi. 2002. A Computationally Efficient Evolutionary Algorithm for Real-Parameter Optimization. *Evolutionary Computation* 10, 4 (12 2002), 371–395.

[28] Luca Di Gaspero, Giacomo Di Tollo, Andrea Roli, and Andrea Schaerf. 2007. Hybrid Local Search for Constrained Financial Portfolio Selection Problems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 4th International Conference, CPAIOR 2007, Brussels, Belgium, May 23-26, 2007, Proceedings*, Pascal Van Hentenryck and Laurence Wolsey (Eds.). Lecture Notes in Computer Science, Vol. 4510. Springer Verlag, Berlin, Heidelberg, 44–58.

[29] Luca Di Gaspero and Andrea Roli. 2009. Flexible Stochastic Local Search for Haplotype Inference. In *Learning and Intelligent Optimization*, Thomas Stützle (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 74–88.

[30] Luca Di Gaspero, Andrea Roli, and Andrea Schaerf. 2008. EasyAnalyzer: an object-oriented frameowrk for the experimental analysis of stochastic local search algorithms. In *Proceedings of the 15th RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (CEUR Workshop Proceedings)*, Marco Gavanelli and Toni Mancini (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 76–90.

[31] Luca Di Gaspero and Andrea Schaerf. 2001. EasyLocal++: An object-oriented framework for the design of Local Search Algorithms and Metaheuristics. In *Proceedings of the 4th Metaheuristics International Conference (MIC-2001)*, Vol. 2. Porto, Portugal, 287–292.

[32] Luca Di Gaspero and Andrea Schaerf. 2002. Writing local search algorithms using EASYLOCAL++. In *Optimization Software Class Libraries*. Springer, Boston, MA, 155–175.

[33] Luca Di Gaspero and Andrea Schaerf. 2003. EasyLocal++: An object-oriented framework for flexible design of local search algorithms. *Software — Practice & Experience* 33, 8 (July 2003), 733–765.

[34] Luca Di Gaspero and Andrea Schaerf. 2006. Neighborhood Portfolio Approach for Local Search applied to Timetabling Problems. *Journal of Mathematical Modeling and Algorithms* 5, 1 (2006), 65–89.

[35] Luca Di Gaspero and Andrea Schaerf. 2007. A Composite-Neighborhood Tabu Search Approach to the Traveling Tournament Problem. *Journal of Heuristics* 13, 2 (April 2007), 189–207.

[36] Luca Di Gaspero and Andrea Schaerf. 2007. EasySyn++: A Tool for Automatic Synthesis of Stochastic Local Search Algorithms. In *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, International Workshop, SLS 2007, Brussels, Belgium, September 6-8, 2007, Proceedings*, Thomas Stützle, Mauro Birattari, and Holger H. Hoos (Eds.). Springer Verlag, Berlin-Heidelberg, Germany, 177–181.

[37] Marco Dorigo and Thomas Stützle. 2004. *Ant Colony Optimization.* The MIT Press.

[38] Johann Dreo, Arnaud Liefooghe, Sébastien Verel, Marc Schoenauer, Juan J. Merelo, Alexandre Quemy, Benjamin Bouvier, and Jan Gmys. 2021. Paradiseo: From a Modular Framework for Evolutionary Computation to the Automated Design of Metaheuristics: 22 Years of Paradiseo. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. Association for Computing Machinery, New York, NY, USA, 1522–1530.

[39] Juan J. Durillo and Antonio J. Nebro. 2011. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software* 42, 10 (2011), 760–771.

[40] Andreas Fink and Stefan Voß. 2002. HotFrame: A heuristic optimization framework. In *Optimization Software Class Libraries*. Kluwer Academic Publishers, 81–154.

[41] Carlos Fonseca and Alexandre Jesus. 2023. SIGEVO Summer School 2023 Modelling Projects. https://github.com/cmfonseca/s3-2023. [Accessed: 2024-04-05].

[42] Alberto Franzin and Thomas Stützle. 2019. Revisiting simulated annealing: A component-based analysis. *Computers & Operations Research* 104 (2019), 191–206.

[43] Fred Glover and Manuel Laguna. 1997. *Tabu Search.* Kluwer Academic Publishers, Boston, MA.

[44] HeuristicLab 2024. https://github.com/heal-research/HeuristicLab [Accessed: 2024-04-05].
[45] Hexaly 2024. https://www.hexaly.com/ [Accessed: 2024-04-05].
[46] John H. Holland. 1992. Genetic Algorithms. *Scientific American* 267, 1 (1992), 66–73.
[47] Holger H. Hoos. 2012. Programming by optimization. *Commun. ACM* 55, 2 (feb 2012), 70—-80.
[48] Scott Kirkpatrick, Daniel Gelatt, and Mario Vecchi. 1983. Optimization by simulated annealing. *Science* 220 (1983), 671–680.
[49] Scott Kirkpatrick, Daniel Gelatt, and Mario P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220, 4598 (1983), 671–680.
[50] Maria Amélia Lopes Silva, Sérgio Ricardo de Souza, Marcone Jamilson Freitas Souza, and Moacir Felizardo de França Filho. 2018. Hybrid metaheuristics and multi-agent systems for solving optimization problems: A review of frameworks and a comparative analysis. *Applied Soft Computing* 71 (2018), 433–459.
[51] 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 (2016), 43–58.
[52] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. 2003. Iterated local search. In *Handbook of metaheuristics*. Springer, 320–353.
[53] Modern GALib 2024. https://github.com/s-martin/galib [Accessed: 2024-04-05].
[54] Federico Pagnozzi and Thomas Stützle. 2019. Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems. *European Journal of Operational Research* 276, 2 (2019), 409–421.
[55] JoséAntonio Parejo, Antonio Ruiz-Cortés, Sebastián Lozano, and Pablo Fernandez. 2012. Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing* 16, 3 (2012), 527–561.
[56] Pierre Schaus, Renaud De Landtsheer. 2016. OscaR User-guide. https://www.info.ucl.ac.be/~pschaus/oscardoc/ [Accessed: 2024-04-05].
[57] Wolfgang Pree. 1994. Meta patterns — A means for capturing the essentials of reusable object-oriented design. In *Object-Oriented Programming*, Mario Tokoro and Remo Pareschi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 150–162.
[58] Aurora Ramírez, Rafael Barbudo, and José Raúl Romero. 2023. An experimental comparison of metaheuristic frameworks for multi-objective optimization. *Expert Systems* 40, 4 (2023), e12672.
[59] Paolo Rinaldin. 2003. JEasyLocal: Refactoring, riprogettazione e sviluppo di un framework orientato agli oggetti per la ricerca locale. Master thesis (in Italian).
[60] Roberto Maria Rosati, Matteo Petris, Luca Di Gaspero, and Andrea Schaerf. 2022. Multi-neighborhood simulated annealing for the sports timetabling competition ITC2021. *Journal of Scheduling* 25, 3 (2022), 301–319.
[61] Andrea Schaerf, Marco Cadoli, and Maurizio Lenzerini. 2000. LOCAL++: A C++ framework for local search algorithms. *Software: Practice and Experience* 30, 3 (2000), 233–257.
[62] Eric O. Scott and Sean Luke. 2019. ECJ at 20: toward a general metaheuristics toolkit. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (Prague, Czech Republic) *(GECCO '19)*. Association for Computing Machinery, New York, NY, USA, 1391–1398.
[63] Jerry Swan, Steven Adriaensen, Mohamed Bishr, Edmund K. Burke, John A. Clark, Patrick De Causmaecker, Juanjo Durillo, Kevin Hammond, Emma Hart, Colin G. Johnson, Zoltan A. Kocsis, Ben Kovitz, Krzysztof Krawiec, Simon Martin, J. J. Merelo, Leandro L. Minku, Ender Özcan, Gisele L. Pappa, Erwin Pesch, Pablo Garcia-Sànchez, Andrea Schaerf, Kevin Sim, Jim Smith, Thomas Stützle, Vo Stefan, Stefan Wagner, and Xin Yao. 2015. A Research Agenda for Metaheuristic Standardization. In *Proceedings of the XI metaheuristics international conference (MIC 2015)*. Agadir, June 7–10, 2015, 1–3. MIC 2015: the XI Metaheuristics International Conference ; Conference date: 07-06-2015 Through 10-06-2015.
[64] Jerry Swan, Steven Adriaensen, Alexander E.I. Brownlee, Kevin Hammond, Colin G. Johnson, Ahmed Kheiri, Faustyna Krawiec, J.J. Merelo, Leandro L. Minku, Ender Özcan, Gisele L. Pappa, Pablo García-Sánchez, Kenneth Sörensen, Stefan Voß, Markus Wagner, and David R. White. 2022. Metaheuristics "In the Large". *European Journal of Operational Research* 297, 2 (2022), 393–406.
[65] Jerry Swan, Steven Adriænsen, Adam D. Barwell, Kevin Hammond, and David R. White. 2019. Extending the "Open-Closed Principle" to Automated Algorithm Configuration. *Evolutionary Computation* 27, 1 (03 2019), 173–193.
[66] Rodrigo F. Toso and Mauricio Resende. 2015. A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software* 30 (01 2015), 81–93.
[67] Tommaso Urli. 2014. *Hybrid meta-heuristics for combinatorial optimization.* Ph. D. Dissertation. Università degli Studi di Udine.
[68] Rob J.M. Vaessens, Emile H.L. Aarts, and Jan K. Lenstra. 1998. A local search template. *Computers & Operations Research* 25, 11 (1998), 969–979.
[69] David Van Bulck, Dries Goossens, and Andrea Schaerf. 2023. Multi-neighbourhood simulated annealing for the ITC-2007 capacitated examination timetabling problem. *Journal of Scheduling* (2023), 1–16. https://doi.org/10.1007/s10951-023-00799-1
[70] Stefan Voß and David L Woodruff. 2002. *Optimization software class libraries*. Springer.
[71] Matthew B. Wall. 1996. *A Genetic Algorithm for Resource-Constrained Scheduling*. Ph. D. Dissertation. MIT Mechanical Engineering Department.
[72] David H. Wolpert and William G. Macready. 1997. No free lunch theorems for optimization. *IEEE Trans. on Evo. Comp.* 1, 1 (1997), 67–82.
[73] Eugenia Zanazzo, Sara Ceschia, Agostino Dovier, and Andrea Schaerf. 2024. Solving the medical student scheduling problem using simulated annealing. *Journal of Scheduling* (2024), 1–14. https://doi.org/10.1007/s10951-024-00806-z