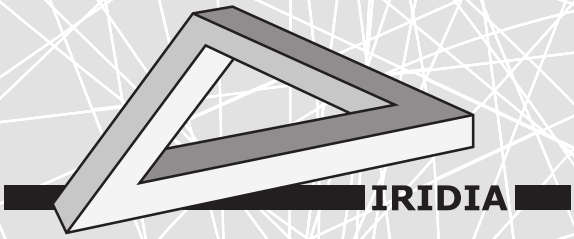

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Revisiting Simulated Annealing: a Component-Based Analysis

A Franzin and T. Stützle

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2018-010

May 2018

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:
IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2018-010

Revisiting Simulated Annealing: a Component-Based Analysis

Alberto Franzin[a], Thomas Stützle[a]

[a]*IRIDIA, Université Libre de Bruxelles (ULB), Belgium*

Abstract

Simulated Annealing (SA) is one of the oldest metaheuristics and has been adapted to solve many combinatorial optimization problems. Over the years, many authors have proposed both general and problem-specific improvements and variants of SA. We propose to accumulate this knowledge into automatically configurable, algorithmic frameworks so that for new applications that wealth of alternative algorithmic components is directly available for the algorithm designer without further manual intervention. To do so, we describe SA as an ensemble of algorithmic components, and describe SA variants from the literature within these components. We show the advantages of our proposal by (i) implementing existing algorithmic components of variants of SA, (ii) studying SA algorithms proposed in the literature, (iii) improving SA performance by automatically designing new state-of-the-art SA implementations and (iv) studying the role and impact of the algorithmic components based on experimental data. We experimentally demonstrate the potential of this approach on three common combinatorial optimization problems, the quadratic assignment problem and two variants of the permutation flow shop problem.

1. Introduction

Metaheuristics are a method of choice when dealing with computationally hard problems from a wide range of application areas [1, 2]. They can be described as problem-independent general rules to follow to derive effective heuristic optimization algorithms. The field of metaheuristics has a long and often successful history that can be traced back to the 1960s and 1970s with the first proposals of evolutionary computation techniques [3, 4, 5, 6] or ideas related to search intensification and diversification [7] that later lead to tabu search [8, 9] or scatter search. Despite the successes, a critical review of the history of the field given in [10] argues that "It is not an exaggeration to claim that the field of (meta)heuristics [...] has yet to reach a mature state". One aspect of this statement is the unfortunate proliferation of dubiously novel methods, generally based on natural metaphors [11, 12] and "high" or "promising" performance claims sometimes backed with poor scientific practices (see e.g. [12, 13]).

The main objective of this work is to propose an alternative way of addressing such issues. Instead of proposing yet another metaheuristic, we aim at exploiting the enormous body of knowledge available in specific metaheuristics and identifying the basic ideas that are available for the design of new variants of the known metaheuristics. For this purpose, we see a metaheuristic algorithm not as a monolithic procedure that is proposed as one block, but as being composed of a set of algorithmic components for each of which a number of different alternative instantiations exist. In other words, we take a component-based view of metaheuristics and we collect many options available in the literature into an *algorithmic framework*, classifying them according to their purpose

Email addresses: afranzin@ulb.ac.be (Alberto Franzin), stuetzle.ulb.ac.be (Thomas Stützle)

for the metaheuristic under concern. From this point of view, algorithm design turns into the task of choosing the right set of basic component from the framework.

In this paper, we build a framework for Simulated Annealing (SA), which is one of the oldest and most studied metaheuristics. In fact, at the time of writing this article, the Scopus bibliographic database indexes more than 6 000 articles with the keyword "Simulated Annealing" in the title, a number that increases to 30 000 if we expand the search to abstracts and keywords. SA also has shown to result in high-performing heuristics for many problems [14, 15]. Over the years, authors have proposed many variants of SA for different problems, offering by now a large number of implementation choices to be taken by an algorithm designer who would like to use SA. We build this framework by taking a component-wise perspective on the design of SA algorithms. The algorithmic components (including alternative algorithm options and numerical parameters) we extract from proposed variants of SA algorithms are classified according to their usage in SA algorithms and alternative choices are offered for each main class of components. This framework is designed and built in such a way that automatic algorithm configuration tools can be directly employed to generate high-performing algorithm variants.

From the point of view of algorithm configuration, each component can be seen as a categorical parameter, whose values are the various options provided in the framework, and each of these components may have associated additional numerical parameters. By choosing the right options and the respective numerical parameters, one can re-instantiate existing algorithms; by choosing different options, instead, it is possible to build new variants. This point of view allows to exploit the recent developments in automatic algorithm configuration, where, given a set of training instances of the problem to be solved, search techniques choose automatically, that is, without manual algorithm designer interaction, the best parameter settings using computer experiments [16]. This task of automated algorithm configuration is supported directly by recent tools such as ParamILS [17], SMAC [18], or irace [19]. In this perspective, our work follows other proposals for the generation of automatically configurable algorithm frameworks that includes work such as Satenstein, a framework for local search algorithms for the satisfiability problem in propositional logic [20, 21], frameworks for multi-objective ACO algorithms [22], ACO algorithms for continuous optimization [23], or multi-objective evolutionary algorithms [24].

This automated process offers at least four advantages. First, it avoids the often applied manual trial-and-error process, which is time-consuming and biased by the personal experience of the algoritm developper. Second, in the framework typically many more algorithm components are made available than even an experienced developer of metaheuristic algorithms may be aware of. In fact, the literature on SA (and many other metaheuristics) is vast and it is virtually impossible to know all possible alternatives. Third, it allows to configure algorithms for a specific computational environment or application context in a transparent (and reproducible) way. Fourth, the data generated during the algorithm configuration process may be further analyzed and so insight into the importance of specific algorithm components can be obtained these data. We experimentally show the advantages of our approach studying SA algorithms for three well-known combinatorial optimization problems, the the quadratic assignment problem (QAP) and the permutation flowshop scheduling problem (PFSP) under the makespan and total completion time objectives.

The paper is structured as follows. In the next section we review SA, and, in Section 3, we describe its component-based formulation and the set of components we have implemented. Section 4 describes the methodology and the experimental setup for the experiments reported in Sections 5 and 6. Additional analysis in given in Section 7 and we conclude in Section 8. Supplementary material for this work is available at [25].

2. Simulated Annealing

In a nutshell, SA is a stochastic local search algorithm that, starting from some initial solution, iteratively explores the neighbourhood of the current solution. It always accepts improving solutions and worsening solutions probabilistically in dependence of the amount of deterioration and a parameter called temperature.

SA is inspired by the work of Metropolis *et al.* [26], who proposed a Monte Carlo integration for solving equations of state of physical systems composed of particles in statistical mechanics. At high temperatures the particles are free to move, and the structure is subject to substantial changes. The temperature decreses over time, and so does the probability for a particle to move, until the system reaches its ground state, the one of lowest energy. Kirkpatrick and co-authors [27], and independently Černy [28], turned these ideas into a heuristic method for tackling combinatorial optimization problems. The physical temperature is translated into a "temperature" parameter, the state of the physical system corresponds to a candidate solution of a problem instance, the ground state corresponds to the globally optimal solution for the instance, and a change of state corresponds to a move to a neighbouring candidate solution.

Let us first introduce the formal notation used in the remainder of this work. Let $s \in S$ be a solution in the set S of all possible candidate solutions and $f : S \to \mathbb{R}$ be the objective function; thus, $f(s)$ is the objective function value of candidate solution s. An optimal solution s^* is a candidate solution for which holds $f(s^*) \leq s \quad \forall f(s) \in S$. With $\mathcal{N}(s)$ we denote the neighbourhood of s. $\Delta(s, s^{'})$ is the objective function difference of two candidate solutions s, $s^{'}$; we will also refer with $\Delta_{i,j}$ to the difference $f(s_j) - f(s_i)$ of objective function values of two candidate solutions in two different instants i and j for brevity. We denote the temperature as T; T_0 and T_f are, respectively, the initial and final temperature, while T_i is the temperature at a generic instant i. Without loss of generality, we assume the objective function to be minimized.

The distinguishing characteristic of SA at its inception was the possibility of probabilistically accepting worsening moves. The most commonly used acceptance criterion is the so-called Metropolis condition, which always accepts a neighboring candidate solution if it is better or equal to the current one; a worse neighboring candidate solution is accepted with a probability of $\exp(\Delta(s, s^{'})/T)$. Hence, a worsening solution is accepted with a probability that depends both on the amount of worsening and the temperature parameter. With an equal worsening of the objective function value, a solution is more likely to be accepted when the temperature is high (that is, typically in the beginning of the search), while when the temperature is low (typically towards the end the search), improving candidate solutions are prioritized. The probabilistic acceptance of worsening moves makes SA able to reach the globally best solution, when certain conditions are met. Several authors have studied these conditions, especially focusing on the cooling scheme, the function that controls the temperature iteration after iteration [29, 30, 14, 31]. Unfortunately, these analysis usually prove the convergence to the global optimum in time tending to infinity making the implications in practice less clear. As in this work we focus on SA from an empirical perspective, we do not delve into theoretical analyses but refer the reader for such to [32, 33, 14, 15, 34, 35] and cited works therein.

There are several reasons that make SA an ideal benchmark for the approach outlined in Section 1. Deriving from a simple idea, in its original formulation it is also a simple algorithm, making it possible to clearly identify its components (see Section 3) and their scope. It does not require complex operations, so its behaviour is easy to understand. The role and the impact of the numerical parameters is well understood: the temperature, transitioning from its initial value to its final one, controls the transition from an initial exploratory behaviour to a final exploitative one; if its values are too high, the algorithm will fail to converge towards good solutions, but for too low values it will

be likely trapped in suboptimal regions, missing the chance to escape. At the same time, the task of choosing the right values (and making the right design choices) is often a tedious, error-prone manual one, so in practice it is difficult to find the best setup.

In the literature there are several algorithms that can be related to SA. For example, keeping the same temperature value throughout the whole execution turns SA into an algorithm known under several names, such as Metropolis Algorithm [36], Generalized Hill Climbing [37], Static Simulated Annealing [38], or simply fixed temperature schemes [39, 40]. Replacing the probabilistic acceptance criterion with a deterministic one, it is possible to generate a new class of local search algorithms, such as the Threshold Acceptance [41, 42], Great Deluge Algorithm and Record-to-Record Travel [43], or the more recent Late Acceptance Hill Climbing [44]. All these variants are described in the next Section. A discussion about the similarities and differences with other metaheuristics can be found in [33, 14]. Outside the optimization field, SA is also akin to the Markov chain Monte Carlo (MCMC) method that is extremely popular in several fields such as machine learning, statistics, physics, or economics [45].

Our analysis is limited to SA as a stand-alone search algorithm. Therefore, we do not consider the use of SA as a component within other hybrid algorithms such as the local search in a memetic algorithms. Also, technology-driven improvements such as parallelization techniques or GPU-based implementations are beyond the scope of this article.

3. Component-based formulation of SA

For our purposes, we divide SA into nine different components, seven of which are algorithm-specific and two are problem-specific. These components define, respectively, how an SA algorithm can be specialized to tackle a specific problem. The two problem-specific components are the construction of an Initial Solution, and the generation of a new candidate solution in the Neighbourhood. While the choice of these two components has an important impact on algorithm performance [33], we delay any discussion about them to the following sections, where the specific problems are introduced and tackled.

We give a generic outline of an SA algorithm in Algorithm 1. The seven components that in our framework define an SA algorithm are:

- the choice of the Initial Temperature (line 3 of Algorithm 1);
- the Stopping Criterion, which determines when the execution is finished (line 4);
- the Exploration Criterion whose purpose is to choose a solution in the Neighbourhood (line 5);
- the Acceptance Criterion, which determines whether the new solution replaces the incumbent one (line 6);
- the Temperature Length, which indicates whether the temperature is updated (line 12);
- the Cooling Scheme, which updates the temperature (line 13);
- the Temperature Restart, the component responsible for resetting the temperature to its original or another, high value (line 15).

While these seven components are the ones particular for SA, they may also be based on problem-specific settings, such as the initial temperature adopted in [46]. We will describe these problem-dependent algorithmic components only if they are used in our experiments.

Algorithm 1: Component-based formulation of SA. In SMALLCAPS the components we have identified for our analysis.

Input: a problem instance π, a NEIGHBOURHOOD $\mathcal{N}$ for the solutions, an INITIAL SOLUTION s_0, control parameters
Output: the best solution s^* found during the search

```
1  best solution s* = incumbent solution ŝ = s_0;
2  i = 0;
3  T_0 = initialize temperature according to INITIAL TEMPERATURE;
4  while STOPPING CRITERION is not met do
5  |  choose a solution s_{i+1} in the NEIGHBOURHOOD of ŝ according to EXPLORATION
   |  CRITERION;
6  |  if s_{i+1} meets ACCEPTANCE CRITERION then
7  |  |  ŝ = s_{i+1};
8  |  end
9  |  if ŝ improves over s* then
10 |  |  s* = ŝ;
11 |  end
12 |  if TEMPERATURE LENGTH is met then
13 |  |  update temperature according to COOLING SCHEME;
14 |  end
15 |  reset temperature according to TEMPERATURE RESTART scheme;
16 |  i = i + 1;
17 end
18 return s*;
```

An SA algorithm starts by taking as input the INITIAL SOLUTION, the NEIGHBOURHOOD, a problem instance π and the control parameters. It proceeds by initializing its internal status, in particular setting a value for the INITIAL TEMPERATURE. Starting from the initial solution, SA iteratively selects one candidate solution in the NEIGHBOURHOOD according to the EXPLORATION CRITERION. The new candidate solution is evaluated against the incumbent candidate solution using the ACCEPTANCE CRITERION; if it also improves over the best solution found so far (global-best), it becomes the new global-best candidate solution. The TEMPERATURE LENGTH component determines whether the temperature parameter has to be updated; if yes, the COOLING SCHEME sets the temperature to its new value. To favour a new phase of exploration, the TEMPERATURE RESTART scheme controls whether the temperature should be reset to a higher value. At each iteration, the STOPPING CRITERION is checked–if met, the algorithm terminates returning the best candidate solution found.

We next describe the options for the seven algorithm-specific components we identified in the literature and which we make available in our framework.

3.1. INITIAL TEMPERATURE *(line 3)*

This component sets an initial value for the temperature parameter. The methods available may take into account some problem instance related information or not. The instance-based schemes can be based either on syntactical information, or on a limited exploration of the search space, typically by a random walk in the search space, from which some statistics are computed. Some of the following schemes also propose a final temperature value related to the initial one, but in

practice there is no mandatory value or rule, and they should be considered just as suggestions. A variation that we apply here is to include a multiplicative scaling user-defined constant k to make the methods more flexible.[1]

Fixed value. The simplest option **IT1** is to choose a fixed initial temperature

$$T_0 = k. \tag{1}$$

Another option **IT2** is to set an initial temperature proportional to the objective function value of the initial candidate solution s_0, as in [47]:

$$T_0 = k \times f(s_0). \tag{2}$$

Random walk-based methods. Other criteria perform a random walk in the search space creating a sequence of candidate solutions $s_0, s_1, s_2, \ldots, s_l$, where l is the length of the random walk. The resulting objective function values $f(s_0)$, $f(s_1)$, $f(s_2)$, $\ldots$, $f(s_l)$ are treated as a time series and the temperature is set as a statistic of the time series. One simple option (**IT3**) is to take a value proportional to the maximum gap between two consecutive candidate solutions as initial temperature, and the minimum non-zero gap as final temperature, as for example in [48]:

$$T_0 = k \times \max_{1 \leq i \leq N} |\Delta_{i,i+1}| \tag{3}$$

$$T_f = k \times \min_{1 \leq i \leq N, >0} |\Delta_{i,i+1}|. \tag{4}$$

As the maximum value of a set can be unrepresentative or highly skewed, we also include (**IT4**) the possibility of choosing a value proportional to the average of the absolute gaps between candidate solutions in the random walk

$$T_0 = k/N \times \sum_{i=1}^{N} |\Delta_{i,i+1}| \tag{5}$$

or a more elaborated scheme, as for example in [49] (**IT5**):

$$T_f = k \times \min_{1 \leq i \leq N, >0} min\Delta_{i,i+1} \tag{6}$$

$$T_0 = T_f + k \times (\max_{1 \leq i \leq N} \Delta_{i,i+1} - T_f)/10, \tag{7}$$

thus relating the initial temperature value to its supposed final one.

As the initial temperature is used to control the initial acceptance probability of worsening moves, the initial temperature can be determined such that it yields a desired initial acceptance probability, as done in [50, 51, 52]. This component (**IT6**) performs a random walk in the search space and computes the value

$$T_0 = |\frac{k \times \Delta_{avg}}{\log p_0}|, \tag{8}$$

which gives an initial probability p_0, where Δ_{avg} is the average gap between two solutions in the random walk and k is a scaling coefficient. Equation 8 is derived from the Metropolis acceptance condition [26, 27].

[1] Here and in the following we enumerate the available options we implemented for ease of later reference. Options for initial temperature are refered to as **ITx**, where **x** is a number. Other references are defined analogously in the text.

Misevicius [53] proposed an initial temperature scheme (**IT7**) for QAP, again based on a random walk in the search space, but more elaborated. It extends **IT5** by taking into account also the average gap in the random walk. **IT7** is defined as

$$T_0 = k \times ((1 - \lambda_1 - \lambda'_1)\Delta_{min} + \lambda_1\Delta_{avg} + \lambda'_1\Delta_{max}) \quad (9)$$

$$T_f = k \times ((1 - \lambda_2 - \lambda'_2)\Delta_{min} + \lambda_2\Delta_{avg} + \lambda'_2\Delta_{max}), \quad (10)$$

where the real-valued weights $\lambda_1, \lambda'_1, \lambda_2, \lambda'_2 \in [0, 1]$ are chosen to satisfy the conditions $\lambda_1 + \lambda'_1 \leq 1$, $\lambda_2 + \lambda'_2 \leq 1$. By choosing $\lambda_1 = 0$, $\lambda'_1 = 0.1$, $\lambda_2 = 0$, $\lambda'_2 = 0$ we obtain **IT5**. Misevicius also uses a simplified version (**IT8**) of his scheme, by setting $\lambda'_1 = \lambda'_2 = 0$. By varying the λ_1 and λ_2 values, the behaviour of the search will differ, resulting in faster or slower cooling as needed.

Problem-dependent schemes. While some of the previous schemes were introduced for some specific problems, they are general. In [46], the authors propose an initial temperature for an SA algorithm for the Permutation Flow-Shop Scheduling Problem (PFSP), that uses specific features of a problem instance. We consider this scheme as the PFSP is one of the problems we use in this work. The scheme **IT9** is defined as

$$T_0 = k \times \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij}/(m \times n), \quad (11)$$

where n is the number of jobs, m is the number of machines, p_{ij} is the processing time of job i on machine j and the scaling coefficient k takes the value of 1/5 in the original work. The PFSP will be described more in detail in Section 4.

3.2. Stopping Criterion *(line 4)*

The stopping criterion controls the termination of the SA algorithm. The schemes can be based either on some predetermined value, or on the actual outcome of the search, terminating the algorithm when continuing to search is deemed too expensive, or too unlikely to improve the current results.

Fixed termination. One possible choice for termination (**SC1**) is a fixed maximum amount of time [52, 47]. Another possible choice (**SC2**) is a fixed number of moves [49]. The search can be terminated also when a certain minimum temperature value has been reached (**SC3**) [46]; such value can be either determined by the chosen Initial Temperature scheme, in case it computes also a final value, or given as input by the user. Other possible criteria include having a maximum number of cooling steps (**SC4**) [54], or a maximum number of temperature restarts (**SC5**) [48].

Adaptive termination. To provide a more flexible environment for the search, other criteria based on the observation of the actual execution have been implemented. One such criterion implemented (**SC6**) is to stop the execution after a fixed number of moves that did not result in accepted solutions. **SC7** terminates the execution as soon as the total acceptance rate falls below a certain threshold; another one (**SC8**) stops the search when the acceptance rate for the last k moves is below a given threshold [50, 51]. Finally, criterion **SC9** stops the algorithm when none of the last k moves found a new best solution. The number of moves to be considered by the adaptive criteria can be expressed either in terms of an absolute value or proportional to the size of the neighbourhood.

Note that except for the termination criterion **SC1**, with the other termination criteria the computation time used by an SA algorithm is not determined a priori but depends on the search progress, making the running time an independent variable.

3.3. Exploration Criterion *(line 5)*

The role of the component Exploration Criterion is to choose one candidate solution to evaluate in the neighbourhood $\mathcal{N}(s)$. The traditional SA [27] uses a random exploration of the neighbourhood, that is, it generates and evaluates a randomly generated neighbour at every step (**NE1**). This is the default behaviour of SA, and the one used in the vast majority of the SA implementations.

In [49], Connolly claims this approach to be inefficient, because for lower temperatures (that is, lower acceptance probabilities, see the next component) potential improvements might be missed, and at the same time it might be difficult to escape local optima. He proposes to compute and evaluate the neighbours in some sequential order (**NE2**) arguing that if no worsening move is accepted at least it is guaranteed that the full neighbourhood is explored and a solution is identified as a local optimum.

In [55], Ishibuchi et al. propose two simple local searches to find a suitable solution in the neighbourhood. In the first one (**NE3**), k solutions are randomly generated in the neighbourhood, and the best one of the set is then elected for the comparison with the current incumbent. This scheme is also modified (**NE4**) in a first improvement fashion, where the first one of the k randomly generated neighbouring solutions that improves the current incumbent is immediately accepted; if no improving solution is found, the best one among them is chosen for evaluation as in the previous case.

3.4. Acceptance Criterion *(line 6)*

This is the component that determines whether the solution s' with cost $f(s')$ generated in the neighbourhood of s with cost $f(s)$ by the Exploration Criterion is accepted. The traditional Metropolis criterion [26, 27] is the best known example for this component. Almost all criteria described here follow one simple pattern: always accept improving solutions and occasionally accept worsening solutions depending on a specific criterion. The acceptance of worsening moves allows to move the search away from the current area of the search space being explored, in the hope of finding regions with better solutions. It is, however, crucial to find a good balance between search intensification and diversification, which is managed by setting appropriately the temperature parameter.

Metropolis-based criteria. The Metropolis condition (**AC1**) [26] is the criterion proposed in the original SA formulation [27]. It always accepts improving moves, and accepts worsening moves with a probability that depends on the increase $\Delta(s', s)$ of the objective function value and the temperature T:

$$p_{Metropolis} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ \exp\left(\Delta(s', s)/T\right) & \text{otherwise.} \end{cases} \tag{12}$$

As the exponential function is relatively expensive from a computational point of view, Johnson *et al.* **AC2** [50] proposed to pre-compute the exponentials for a sequence of values in the interval where $\Delta(s', s)/T$ results in probabilities between 1 and ~ 0.0067. Worsening moves that entail probabilities lower than this value are immediately discarded as too bad, and therefore their acceptance too unlikely. The actual values during the search are mapped to the closest values in this array. In their experiments, they estimate a saving of 1/3 on the total runtime.

Chen and Hsieh [56] proposed a bounded version **AC3** of the Metropolis criterion, that rejects

a move whose solution quality is worse with respect to the incumbent by a given parameter ϕ_{BM}:

$$p = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ \exp\left(-\Delta(s', s)/T\right) & \text{if} \quad f(s) < f(s') \leq f(s) \times \phi_{BM} \\ 0 & \text{if} \quad f(s') > f(s)s \times \phi_{BM}. \end{cases} \tag{13}$$

Another acceptance criterion was proposed in [57] (**AC4**) as part of the Generalized Simulated Annealing (GSA) variant. It includes (a power of) the cost of the currently accepted solution in the probability calculation. The formula proposed is

$$p_{GSA} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ \exp\left(-\beta f(s')^g \Delta(s', s)\right) & \text{otherwise,} \end{cases} \tag{14}$$

where β and g are control parameters. GSA makes no explicit use of temperature, but following to their notation the original SA employs $\beta = 1/T$, $g = 0$.

Geometric criterion. A criterion proposed in [58] (**AC5**) always accepts an improving solution, and accepts a worsening solution with a probability that decreases in a geometric way[2]:

$$p^k_{Geom} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ p_0 \times r^{k-1} & \text{otherwise,} \end{cases} \tag{15}$$

where p_0 is the initial acceptance probability, $r < 1$ is the reducing factor, and k is the number of update steps. This criterion is the first occurrence of an acceptance for SA algorithms that does not use the temperature in the evaluation of a solution. In fact, it is rather related to the randomized iterative improvement algorithm as defined in [1].

Deterministic criteria. While the probabilistic acceptance of worsening moves is the distinctive feature of SA, in [59, 41] and independently in [42] and [60], some authors have questioned whether the stochasticity introduced by the acceptance criterion is really necessary to obtain good results. In both works, the authors have proposed a deterministic version of the Metropolis criterion, called Threshold Accepting (TA, **AC6**), which accepts every worsening solution whose difference in objective function value is lower than a threshold $\bar{\phi}$:

$$p_{TA} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq \bar{\phi} \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

$\bar{\phi}$ is a parameter whose value decreases during the search process, just as the temperature in SA. In [42] the authors do not provide any guidance in how to initialize and update $\bar{\phi}$, while the authors of [41] explicitly use SA terminology such as "temperature" and "cooling".

Starting from Threshold Accepting, Dueck in [43] proposed two alternative deterministic acceptance criteria, under two different metaphors. The first one (**AC7**, originally called Great Deluge Algorithm – GDA) accepts every move leading to a solution with objective function value $f(s)$ less than a threshold $\bar{\phi}^k$, therefore moving away from the idea of comparison between solutions:

$$p^k_{GDA} = \begin{cases} 1 & \text{if} \quad f(s) \leq \bar{\phi}^k \\ 0 & \text{otherwise,} \end{cases} \tag{17}$$

[2]We describe this component as proposed in the original paper, even though this formulation combines the acceptance criterion with the Cooling Scheme.

with $\bar{\phi}^{k+1} = \bar{\phi}^k - \lambda$, where λ is a fixed parameter.

The second criterion (**AC8**, in the original work, Record–to–Record Travel – RTR) accepts only solutions whose value is not larger than that of the best solution found so far plus a threshold γ (note that, differently, TA compares the new solution with the current one, which might already not be the best one found):

$$p_{RTR} = \begin{cases} 1 & \text{if} \quad \phi \leq f(s^*) + \gamma \\ 0 & \text{otherwise,} \end{cases} \tag{18}$$

where γ is a fixed parameter.

A more recent work [44, 61], Burke and Bykov propose another deterministic criterion called Late Acceptance Hill Climbing (LAHC, **AC9**). The idea of LAHC is to use the history of the search, by comparing the candidate solution also with an incumbent solution of the past. LAHC therefore can accept worsening moves, but it cannot accept a solution whose objective function value is not at least as good as the one of another solution already accepted. The acceptance of a solution s is therefore controlled according to the formula

$$p^l_{LAHC} = \begin{cases} 1 & \text{if} \quad f(s_l) \leq \max\{f(s_{l-1}), f(s_{l-\kappa})\} \\ 0 & \text{otherwise,} \end{cases} \tag{19}$$

where $f(s_l)$, $f(s_{l-1})$ and $f(s_{l-\kappa})$ are, respectively, the cost of the solution at move l, $l-1$, $l-\kappa$, for a fixed κ, which is a parameter of LAHC.

The baseline for comparisons, and simplest deterministic acceptance criterion fitting in the algorithmic outline given in Algorithm 1 is however the basic acceptance of only improving solutions (**AC10**), discarding worsening moves:

$$p_{det} = \begin{cases} 1 & \text{if} \quad \Delta(s', s) \leq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

The effect of this choice is turning SA into a Hill-Climbing search [62], with its drawback of quickly getting stuck in local optima.

3.5. Cooling Scheme *(line 13)*

The cooling scheme is the component that governs the temperature updates, that is, it computes the temperature value T_{i+1} at instant $i+1$ as a function of the previous value T_i at instant i. The default desired behaviour in SA is a monotonic decrease of the temperature, making the acceptance of worsening solutions more and more unlikely. Since the inception of SA, cooling schemes have been the most studied components, both from a theoretical and an experimental point of view. Many schemes that we review adhere to this decreasing behaviour; the option of raising the temperature is governed by the Temperature Restart component. The possibility of having multiple moves evaluated at the same temperature is defined by the Temperature Length scheme. In the literature there are, however, some non-decreasing schemes, that we also consider in this study.

Geometric schemes. In the original SA paper [27], the authors propose two decreasing geometric schemes, (**CS1**)

$$T_{i+1} = \alpha \times \beta^{T_i} \tag{21}$$

and (**CS2**),

$$T_{i+1} = \alpha \times T_i \tag{22}$$

where $0 < \alpha, \beta < 1$ are constant control parameters. Typically, these parameters are set to high values (e.g. ≥ 0.9), implying a slow decrease of the temperature.

Logarithmic schemes. In [63], the authors use a logarithmic cooling scheme (**CS3**)

$$T_{i+1} = \frac{a}{\log(b+i)} \tag{23}$$

with $b = 1$ in the original work. We also implement the scheme **CS4** [64]

$$T_{i+1} = \frac{a}{b + \log i}. \tag{24}$$

Lundy-Mees and variants. A popular cooling scheme is the one proposed by Lundy and Mees in [29] (**CS5**), in which

$$T_{i+1} = \frac{T_i}{a + b \times T_i}. \tag{25}$$

a and b have to be chosen as $a + b \times T_i > T_i$ to guarantee final convergence.

Connolly in [49] develops a variant of the Lundy-Mees scheme called *Q8-7* (for: seventh variant of the eighth scheme tested) (**CS6**) for the QAP. It is a two-step scheme, that initially decreases the temperature (using the Lundy-Mees formula (25)), until too many consecutive moves m are discarded. Then, the next move is accepted, the cooling is stopped and the temperature is set to the value at which the best solution was found. Connolly sets parameters as $a = 1$ and b, m in dependence of the initial and final temperature and the size of the neighbourhood.

Another scheme (**CS7**) is proposed in [65] and uses

$$T_{i+1} = \frac{a}{1 + b \times T_i} \tag{26}$$

with $a = b = 1$ in the original formulation.

Quadratic schemes. Andersen, Vidal and Iversen [66] developed a quadratic cooling scheme (**CS8**) for a network design problem. The formula proposed is

$$T_{i+1} = a \times K^2 + b \times K + c, \tag{27}$$

$$a = \frac{T_0 - T_f}{I^2}, \tag{28}$$

$$b = 2 \times \frac{T_f - T_0}{I}, \tag{29}$$

$$c = T_0, \tag{30}$$

where K is the current iteration, T_0 the initial temperature, $T_f = 0$ the final temperature and I the maximum number of iterations.

Arithmetic scheme. Another simple cooling scheme proposed in [43] uses an arithmetic decrease of the temperature value (in this case, of the threshold):

$$T_{i+1} = T_i - a, \tag{31}$$

for some fixed value a. The implementation of this criterion (**CS9**) requires a more careful control about the update, as the temperature value cannot drop below zero. We therefore choose to implement the criterion according to

$$T_{i+1} = \max\{T_i - a, 0\}. \tag{32}$$

Non-decreasing schemes. Since the inception of SA, various authors have studed variants that keep the same temperature values throughout the whole search [67, 31, 60, 36, 68, 39, 40, 38], under different perspectives: SA variants, theoretical studies about convergence behaviours, different algorithmic paradigms; and different names, such as Metropolis algorithm, Static SA, Generalized Hill Climbing, Probabilistic Iterative Improvement or simply fixed-temperature SA. All these works essentially (re)propose and study the scheme (**CS10**):

$$T_{i+1} = T_i = T_0 \; \forall i. \tag{33}$$

Here, T_0 is a supposedly *optimal* temperature value that can obtain superior results with respect to cooling schemes that reduce the temperature parameter. We can also consider the *Q8-7* cooling scheme **CS6** of [49] as the first practical application of a fixed temperature value that is discovered by the algorithm at runtime.

As this method is dependent on good initial settings, a more robust scheme (**CS11**) sets a temperature band

$$[T_0, a \times T_0], \tag{34}$$

$a > 0$, and, at each update, randomly chooses a temperature in that interval.

Another scheme is the Old Bachelor Acceptance (OBA) [69]. It was originally proposed as a variation of Threshold Acceptance, which is discussed as a special case; it lowers the temperature if a solution is accepted, and raises it if the candidate solution is discarded. While OBA is conceived to be used with **AC6**, it can be paired with any acceptance criterion, and we describe it here as such, considering the two variants presented in the original paper. OBA1 (**CL12**) adjusts the temperature "symmetrically" according to the formula

$$T_{i+1} = \begin{cases} T_i + ((\text{age}/a)^b - 1) \times \Delta \times (1 - i/M)^c & \text{if} \quad s_i \text{ is discarded} \\ T_i - ((\text{age}/a)^b - 1) \times \Delta \times (1 - i/M)^c & \text{if} \quad s_i \text{ is accepted,} \end{cases} \tag{35}$$

where a, b, c are control parameters, M is the total number of moves, Δ is the granularity of the update and *age* is the number of consecutively rejected moves. OBA2 (**CL13**) instead uses a "steepest descent, mildest ascent" strategy [9] that makes acceptance of a solution more likely in the first d moves after an accepted move:

$$T_{i+1} = \begin{cases} T_i + (\Delta/d) \times (1 - i/M) & \text{if} \quad s_i \text{ is discarded} \\ T_i - \text{count} \times \Delta \times (1 - i/M) & \text{if} \quad s_i \text{ is accepted,} \end{cases} \tag{36}$$

where d and *count* are control parameters, with

$$count = \begin{cases} count + 1 & \text{if} \quad \text{age} < d \\ 1 & \text{if} \quad \text{otherwise.} \end{cases} \tag{37}$$

3.6. Temperature Length *(line 12)*

This component controls the number of moves L that are evaluated at a certain temperature.

Fixed temperature length. The options in this category include the following. The first is to update the temperature after a fixed number of moves (**TL1**):

$$L = k \tag{38}$$

with a value of $L = 1$ meaning that the temperature is updated after every move. The second is to update after a number of moves proportional to the size of the neighbourhood $\mathcal{N}(s)$ (**TL2**):

$$L = k \times |\mathcal{N}(s)|. \tag{39}$$

or proportional to the square of neighbourhood size (**TL3**) [54]:

$$L = k \times |\mathcal{N}(s)|^2. \tag{40}$$

Also the size n of the problem instance (**TL4**) is used:

$$L = k \times n \tag{41}$$

e.g. in [47], or its square (**TL5**) [49]:

$$L = k \times n^2. \tag{42}$$

Adaptive temperature length. Instead of fixed temperature lengths, it is possible to set these depending on the search progress. Abramson [70] updates the temperature after a certain number of accepted moves (**TL6**). In [54] the authors combine this approach with a maximum number of total moves at a given temperature (that might be fixed or proportional to $|\mathcal{N}(s)|$), to avoid spending too many moves at a certain temperature (**TL7**).

Variable temperature length. Other proposals [71, 72] update the temperature length according to some functions, to compensate the increased strictness in accepting worsening moves at low temperatures with an increased number of evaluations. We test arithmetic (**TL8**), geometric (**TL9**), logarithmic (**TL10**) and exponential (**TL11**) updates for temperature lengths, respectively:

$$L_{i+1} = L_i + k \tag{43}$$

$$L_{i+1} = k \times L_i \tag{44}$$

$$L_{i+1} = \frac{k}{T_i} \tag{45}$$

$$L_{i+1} = L_i^{1/\alpha}, \tag{46}$$

where k and $0 < \alpha < 1$ are fixed numerical parameters and T_i is the temperature at iteration i.

3.7. Temperature restart (line 15)

As most cooling schemes decrease the temperature, it eventually happens that the acceptance of worsening solution is very rare, resulting in a Hill-Climbing type behavior (see also the discussion in Section 3.4). Therefore, authors have proposed to reset the temperature to the initial or another level once it reaches a critically low value [67], allowing in this way effective escapes from local optima. The reset of the temperature to its initial value is simply called *temperature restart*, while setting the temperature to a possibly different value, is called *reheating*. Of course, it is also possible to choose to never reset the temperature value (**TR1**).

Restarting, fixed settings. The most immediate option is to reset the temperature value to the initial one, once some conditions are met. For example, once it reaches a minimum absolute value (**TR2**), when it reaches a certain percentage of its initial value (**TR3**), after a certain number of moves (**TR4**, this number being fixed, proportional to the size (or its square) of the neighbourhood) or after a certain number of cooling steps (**TR5**).

Restarting, adaptive settings. In this category, options are to restart when the overall acceptance rate falls below a certain threshold (**TR6**), when the acceptance rate of the last l moves is below a certain threshold (**TR7**), or when the search is not progressing (no accepted moves in the last l iterations, **TR8**).

Reheating. Alternatively to restarting the temperature, it is also possible to set it to a higher value

$$T_{i+1} = \frac{T_i}{k}, \tag{47}$$

$0 < k < 1$. We can perform this operation once the acceptance rate falls below a given threshold, overall (**TR9**) or in the last k moves (**TR10**), or the search has not accepted any new solution in the last k moves (**TR11**). We also consider reheating after a certain number of moves (**TR12**, again this number can be fixed or proportional to the size (or its square) of the neighbourhood) or cooling steps (**TR13**). A different version of reheat, called Enhanced Reheat (**TR14** [73]) performs a reheat according to the usual formula $T_{i+1} = T_i/k$, but at every reheating the parameter k gets reduced by a constant value ϵ. To prevent k to become negative, we actually update k using the formula $\max\{\epsilon, k - \epsilon\}$. The idea is to restart the search every time at a higher temperature, to increasingly push the algorithm towards an explorative behaviour.

Another option for reheating is to set the temperature value not to its original value or to another "generic" higher value, but to the temperature at which the best solution has been found, assuming that value being a good one in terms of acceptance probability. We can reset the temperature to this supposedly "optimal" value when the acceptance rate drops below a threshold (**TR15**) or when k consecutive moves have been rejected (**TR16**).

4. Material and method

We have collected the algorithmic components described in Section 3 into an algorithmic framework, from which it is possible to instantiate, following the outline of Algorithm 1, a fully working algorithm. This outline also defines the ways the algorithmic components can be combined. The implementation itself is done in the EMILI framework [74], which aims at a flexible combination of algorithm components that makes it particularly amenable to automatic configuration.

In the following, we illustrate the possible uses of the framework at various levels of automatic configuration ranging from no configuration to full configuration of the framework as

Level 1: instantiate known SA algorithms for algorithm comparisons;

Level 2: tune known SA algorithms for fair comparisons;

Level 3: automatically configure the full SA framework.

These three levels correspond to different degrees of advancement in algorithm comparisons. In fact, in most current publications on metaheuristics, a new algorithm is compared to previously proposed ones using only published results. Following Level 1, we instantiate all algorithms from a same code base and execute them in a same environment; such procedure already removes (noise) factors such as different implementation languages, implementation skills, and different computing environments. Level 2 advances over Level 1 by tuning the numerical parameters of each algorithm by automatic algorithm configuration techniques, reducing the side-effect of uneven tuning of the methods or using parameters fine-tuned for possibly other experimental conditions (e.g. short versus long computation times). Level 3 improves over Level 2 by a modern view on metaheuristic algorithm engineering as seeing these algorithms composed of different algorithm components [75, 16].

In fact, given the possibility of generating new, previously unseen SA algorithms potentially better performance may be reached. From an automatic configuration perspective, this is obtained by appropriate choices for the categorical parameters through automatic configuration. For the automatic configuration, we consider two setups. The first one configures the algorithm to reach the best solution quality within a given maximum computation time. While this is the most common setup when comparing metaheuristic algorithms, it has the disadvantage that minor changes in the available computation time or the computing environment (slower or faster machines) may have a major impact on algorithm performance. Therefore, in a second setup we automatically configure SA algorithms for anytime behavior, which tries to obtain as good solutions as early as possible during the search [76]. To do so, we follow the methodology proposed in [77].

The data obtained during the automatic configuration process can be exploited to obtain insight into the importance of the various algorithm components and the numerical parameters. As a final step, we analyze the importance of the algorithm components from the data that are recorded during the configuration with irace by training a random forest model [78].

4.1. Experimental setup

As mentioned above, we implemented the SA components within the EMILI framework [74]. As the automatic algorithm configuration tool we use irace [79, 19], which is an R implementation of the Iterated Racing algorithm [80, 81]. irace begins with a set of uniformly sampled candidate parameter configurations and tests them on a set of training instances. Configurations that are statistically worse get discarded during this process to save computational budget for the most promising candidates and to evaluate them on more instances. The best configurations are then used as seeds to sample new candidates, with a distribution skewed around the best performing ones. This process is iterated until the configuration budget is exhausted. The final configurations returned are the ones that performed best during the training phase. For the configurations to generalize to production environments or simply to an independent test set, it is responsibility of the user to provide a set of training instances that is representative for the desired use case. The random forest model is computed using the `ranger` `R` package [82]. For our experiments we consider two types of problems, the Quadratic Assignment Problem (QAP), and the Permutation Flow Shop Problem (PFSP), details on which we present below. We tune the numerical parameters 30 times, with a budget of 2000 experiments per tuning; the tuning of the whole framework is instead done 15 times, with a budget of 60000 experiments due to the much larger number of parameters. The tuning for anytime behaviour is performed three times. The possible values for the numerical parameters are reported in the Supplementary Material. The maximum time limit depends on the problem, and is specified in the following sections. The comparison between algorithms is always performed on the same set of random seeds (hence, using the same number of algorithms tuned or generated). All experiments have been run on a machine equipped with two Intel Xeon E5-2680 v3 CPUs running at 2.5GHz, with 16MB cache and 2.4 GB of RAM available per algorithm execution. Each algorithm execution is single-thread.

4.2. QAP setup

The QAP is an assignment problem that models a variety of real world problems [83, 84]. Each QAP instance of size n has n *facilities* and n *locations*, and associated costs commonly referred to as *flow* between two facilities i and j, f_{ij}, and *distance* between two locations k and l, d_{kl}. An assignment of facilities to locations can be represented by a permutation π, where $\pi(i)$ gives the location to which facility i is assigned. The goal in the QAP is to find a permutation π that

minimizes the objective function

$$\sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij} d_{\pi(i)\pi(j)}. \tag{48}$$

As per the problem-specific components, the initial solution is a permutation that is generated uniformly at random, while the neighbourhood is the exchange neighbourhood defined as

$$\mathcal{N}(\pi) = \{\pi' \mid \pi'(j) = \pi(h) \wedge \pi'(h) = \pi(j) \wedge \forall : l \notin \{j, h\}\, \pi'(l) = \pi(l)\} \tag{49}$$

for a solution π. The size of this neighbourhood is $n(n-1)/2$.

The QAP is a "difficult" NP-hard problem for which exact solutions can be found only for instances of small size: apart from few exceptions for very specially structured instances [85], instances of size $n = 40$ are often already out of reach for exact methods. We consider two sets of QAP instances. One is composed by instances whose data matrices are generated uniformly at random [47], and one where the matrices are generated randomly according to an Euclidean structure, closer to real-life instances [86]. When no ambiguity can arise we will refer to these two sets as random and structured instances, respectively. Each instance set is composed by 300 instances, equally divided in sizes 60, 80 and 100. Of each size, 50 instances are reserved for the training set to be used during the configuration phase and 50 instances as the independent test set for the evaluation of the configured algorithms. Anytime behaviour is evaluated also on larger random and structured instances of size 500 from the same benchmark [47]. Unless otherwise specified, the runtime limit for QAP is 10 seconds.

4.3. PFSP setup

The PFSP is a scheduling problem that arises in various industrial environments [87, 88, 89]. It is very well studied with many variants existing in the literature. We consider two of the most common variants, namely the PFSP under the *Makespan* objective (PFSP-MS) [90] and under the *Total Completion Time* objective (PFSP-TCT) [91, 92]. More formally, in the PFSP a set of n jobs have to be ordered for execution on a set of m machines, using the same execution order on all machines. Each job i takes p_{ij} units of time for processing on machine j. In the basic formulation of the PFSP, all jobs are ready for execution at time 0 and no concurrency or pre-emption is allowed. A solution of the PFSP is a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ of the n jobs. The PFSP-MS requires to minimize the makespan C_{max}, which is the completion time of the last job executed ($C_{max} = C_{\pi(n),m}$, where $C_{i,m}$ is the completion time of job i on the last machine m). The objective of the PFSP-TCT is to minimize the sum of the jobs' completion times, given by $\sum_{i=1}^{n} C_{i,m}$.

For both objectives, the initial solution is generated using the NEH heuristic [93]. Unless specified otherwise, the neighbourhood is the insert neighbourhood where a move (j, k) consists in selecting the element $\pi(j)$ in position j of the permutation π and inserting it in position $k \neq j$, resulting in a permutation $\pi' = [\pi(1), \ldots, \pi(j-1), \pi(j+1), \ldots, \pi(k), \pi(j), \pi(k+1), \ldots, \pi(n)]$ if $j < k$ and $\pi' = [\pi(1), \ldots, \pi(k-1), \pi(j), \pi(k), \pi(k+1), \ldots, \pi(j-1), \pi(j+1), \ldots, \pi(n)]$ if $j > k$. The size of the insert neighbourhood is $n(n-1)$.

The training set is composed by 40 randomly generated instances with sizes between 50 jobs and 20 machines to 250 jobs and 50 machines [94]. The test set is the popular Taillard benchmark [95], consisting in 120 instances divided into 12 classes of 10 instances each, with sizes going from 20 jobs and 5 machines to 500 jobs and 20 machines. As additional benchmark for the anytime behaviour we use instances of size 800×60 from [96]. The runtime limit is based on the instance size and is computed as $(n \times m \times 0.015)/2$ seconds.

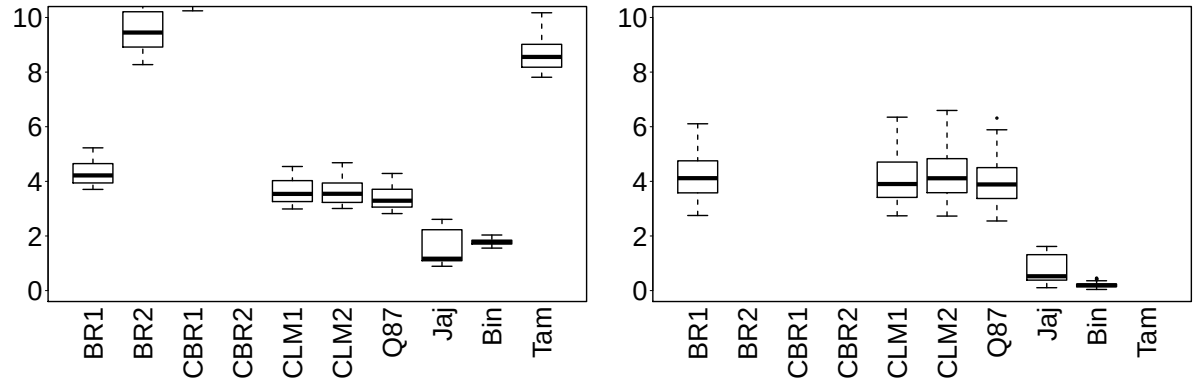

Figure 1: Average Relative Percentage Deviation (ARPD) from the best known solutions obtained by the algorithms in their original settings on random (left plot) and structured instances (right plot).

5. SA algorithms for the QAP

Attempts to solve the QAP with Simulated Annealing can be traced back at least to 1984 [48]. In the following years SA remained a popular choice for the QAP, and was often compared with Tabu Search, without any clear consensus in the scientific community about which method is the most effective [97, 98, 99, 100]. In what follows we list SA implementations proposed for the QAP or closely related problems.

The first two schemes, `BR1` and `BR2`, we implement are proposed as early as in 1984 by Burkard and Rendl [48]. We consider also the SA of Burkard and Rendl as described by Connolly in [49] for comparing the results of his experiments. Connolly [49] proposes several versions of SA for QAP: two of them employ the cooling scheme of Lundy and Mees, while the third version features the alternative cooling scheme Q8-7. Two other SA algorithms for the QAP were proposed by Jajodia *et al.* [54] and Tam [52] in 1992. The last SA for the QAP that we evaluate is by Hussin *et al.* [47]. A synopsis of how these implementations can be described in terms of their components is given in Table 3.

As a first step at Level 1 in our analysis, we instantiate these algorithms in our framework using the parameter settings proposed in the original papers. Figure 1 gives the results on the test sets in terms of the Average Relative Percentage Deviation (ARPD) from the best known solutions, while Table 1 reports the ranking of the algorithms. Algorithms `Bin` and `Jaj` obtain the lowest ARPD values (less than 2% ARPD on random instances, and less than 1% on the structured ones), and are significantly better than the other SA algorithms on structured and random instances when using default algorithm parameter settings. Maybe surprisingly, some algorithms (`BR2`, `CBR1`, `CBR2` and `Tam`) report ARPDs around 10% or higher, much worse than a simple first improvement scheme that yields an ARPD of about 5%. An explanation for this fact may be that many of these algorithms have been proposed when a much lower computational power was available[3], and have been tuned manually for, by current standards, very small instances; apparently, the parameter settings do not scale to the larger instances we consider here.

[3]The processing power alone for mid-range CPUs have increased roughly four orders of magnitude in the last 30 years: an Intel i486 in 1989 measured 8.7 MIPS at 33MHz, while an Intel i7 7500U in 2016 scores 49360 MIPS at 2.7GHz, (with a ratio of 5673). Source: https://en.wikipedia.org/wiki/Instructions_per_second#Timeline_of_instructions_per_second

Table 1: Results of the Friedman rank sum test for the ten algorithms in their default settings on the QAP instances. Algorithms are ranked according to their results. Δ_R is the minimum rank-sum difference that indicates significant difference from the best one. Algorithms in boldface are significantly better than the following ones.

Instances	Δ_R	Algorithm ranking
Random	10.92	**Jaj** (0), Bin (50), Q87 (251), CLM2 (453), CLM1 (496), BR1 (700), Tam (850), BR2 (1000), CBR1 (1150), CBR2 (1300)
Structured	21.3	**Bin** (0), Jaj (148), Q87 (391), CLM1 (486), BR1 (607) CLM2 (612), Tam (899), BR2 (1049), CBR1 (1199), CBR2 (1349)

Table 2: Results of the Friedman rank sum test for the ten algorithms with ten seconds of runtime on the QAP instances. Algorithms are ranked according to their results. Δ_R is the minimum rank-sum difference that indicates significant difference from the best one. Algorithms in boldface are significantly better than the following ones.

Instances	Δ_R	Algorithm ranking
Random	26.44	**Bin** (0), Jaj (48), CBR1 (166), Tam (286), CLM2 (641) BR1 (650), CLM1 (684), Q87 (925), BR2 (1101), CBR2 (1249)
Structured	17.82	**Bin** (0), Jaj (250), CBR1 (296), Tam (350), BR1 (599) CLM1 (801), CLM2 (847), Q87 (1049), BR2 (1216), CBR2 (1332)

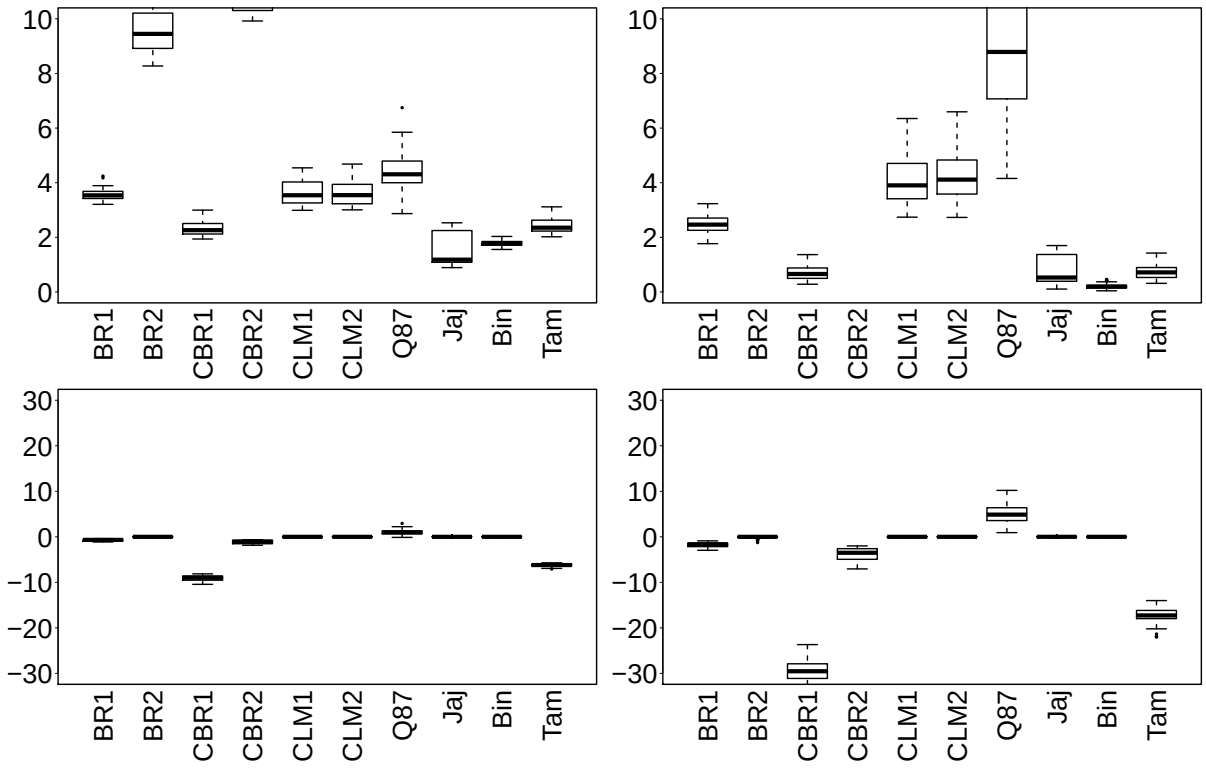

Figure 2: Average Relative Percentage Deviation (ARPD) from the best known solutions obtained by the algorithms with ten seconds of execution time on random (left plot) and structured instances (right plot). In the second row we show the difference with respect to the results of Figure 1; a boxplot below 0 means that the performance improve.

Algorithm	Temperature components	Exploration
`BR1`	**IT3**($l = 10^4$, $k = 1$) **AC1** **CS2**($\alpha = 0.5$) **TL4**($k = 2$) **TR4**($\alpha = 1$) **SC5**($n = 10$)	**NE1**
`BR2`	**IT3**($l = 10^4$, $k = 1$) **AC1** **CS2**($\alpha = 0.5$) **TL9**($k = 1.1$) **TR4**($\alpha = 1$) **SC5**($n = 1$)	**NE1**
`CBR1`	**IT3**($l = 10^4$, $k = 1$) **AC1** **CS2**($\alpha = 0.99$) **TL5**($k = 2$) **TR1** **SC2**($n = 50 \times \|N(s)\|$)	**NE1**
`CBR2`	**IT3**($l = 10^4$, $k = 1$) **AC1** **CS2**($\alpha = 0.99$) **TL9**($k = 1.1$) **TR1** **SC2**($n = 50 \times \|N(s)\|$)	**NE1**
`CLM1`	**IT5**($l = 10^4$, $k = 1$) **AC1** **CS5**(a,b based on the termination) **TL1**($k = 1$) **TR1** **SC2**($n = 50 \times \|N(s)\|$)	**NE1**
`CLM2`	**IT5**($l = 10^4$, $k = 1$) **AC1** **CS5**(a,b based on the termination) **TL1**($k = 1$) **TR1** **SC2**($n = 50 \times \|N(s)\|$)	**NE2**
`Q87`	**IT5**($l = 10^4$, $k = 1$) **AC1** **CS6**(a,b,m based on the termination) **TL1**($k = 1$) **TR1** **SC2**($n = 50 \times \|N(s)\|$)	**NE2**
`Jaj`	**IT6**($l = 10^4$, $k = 1$, $p = 0.8$) **AC1** **CS2**($\alpha = 0.9$) **TL6**($n = 10$, max $= 100$) **TR1** **SC4**($n = 100$)	**NE1**
`Bin`	**IT2**($k = 0.005$) **AC1** **CS2**($\alpha = 0.9$) **TL4**($k = 100$) **TR2**($n = 1$) **SC1**($n = 10s$)	**NE2**
`Tam`	**IT6**($l = 10^4$, $k = 1$, $p = 0.6$) **AC1** **CS2**($\alpha = 0.95$) **TL2**($k = 2$) **TR1** **SC1**($n = 50 \times \|N(s)\|$)	**NE1**

Table 3: Default settings for the ten algorithms of Section 5.

Table 4: Results of the Friedman rank sum test for the ten algorithms after tuning their numerical parameters on the QAP instances, including the algorithms automatically generated (`All`). Algorithms are ranked according to their results. Δ_R is the minimum rank-sum difference that indicates significant difference from the best one. Algorithms in boldface are significantly better than the following ones.

Instances	Δ_R	Algorithm ranking
Random	23.71	**All** (0), `Jaj` (164), `CBR2` (323), `Tam` (417), `BR2` (685), `CBR1` (851) `BR1` (858), `Bin` (902), `Q87` (1230), `CLM2` (1320), `CLM1` (1500)
Structured	39.32	**All** (0), `Bin` (79), `BR1` (271), `Tam` (547), `Jaj` (587), `CBR1` (653) `CBR2` (660), `BR2` (867), `Q87` (1209), `CLM2` (1212), `CLM1` (1428)

This comparison is however unfair: `Bin` and `Jaj` run for the maximum allowed time of ten seconds, while the other algorithms in their default setting use a termination based on a maximum number of moves, resulting here in computation times of fractions of seconds. Hence, as a second step still on Level 1, we allow all algorithms to run for the same maximum ten CPU seconds. The results are reported in Figure 2 where on the top the ARPDs are given and on the bottom the differences to the original settings. In Table 2 we report the rankings. Surprisingly, only few algorithms (`CBR1`, `CBR2` and `Tam`, and `BR1` on the structured instances) benefit from the higher runtimes; probably because the original parameter settings force a (nowadays) fast convergence, independent from the computation time. Conversely, `Q87` worsens its performance, notably on structured instances. This is due to a poor specification of a numerical parameter in the *Q8-7* cooling scheme, the threshold of consecutive rejected moves that triggers the fixed temperature phase. The value of this parameter is originally set according to the total number of solutions evaluated, the original stopping criterion for the algorithm. Consequently, we scaled this parameter up by estimating the number of moves that can be generated in the given time. While the idea of the cooling scheme makes sense, the formula is not robust to a much larger number of moves.

The results for increased computation time indicate that a re-configuration of the algorithm parameters is necessary for a more fair and meaningful comparison. This we do in Level 2 of our analysis, where we tune the algorithms' numerical parameters. In Figure 3 we report in the top row the quality of the solutions returned by the ten algorithms after the tuning of numerical parameters considering a computation time limit of 10 CPU seconds; in the bottom row, the improvement over the non-tuned ten seconds execution of Figure 2. All algorithms now improve their performance; only for `Bin` on the structured instances no statistically significant difference (p-value of 0.698) is observed, as apparently it was already automatically tuned in its original formulation; there is some difference on the random instances instead, probably due to a different tuning setup with respect to the original work.

Almost all the algorithms report very good results, meaning that the original algorithmic ideas were seemingly good, but the originally chosen parameter settings did not generalize to settings (e.g. instances, computation times) different from the ones considered in the original papers. The rankings for the two different instance classes differ more than in the previous two cases, reflecting the difference in the two scenarios. The algorithms can be divided in two groups based on their performance after the tuning, with the ones using the Geometric cooling scheme outperforming `CLM1`, `CLM2` and `Q87`, which instead use the Lundy-Mees cooling scheme or its variant *Q8-7*. Re-tuning `CLM1`, `CLM2` and `Q87` with a higher budget of 5000 experiments does not result in any improvement on the structured instances, and only a slight improvement for `CLM1` and `Q87` on the random instances, though not sufficient to match the results obtained by the algorithms using the Geometric cooling scheme.

The tuning also allows us to observe the "potential" of the algorithms and their components. The case of CLM1 and CLM2 is very interesting in this regard. The only difference between these two algorithms is the neighourhood exploration, with respectively a random and a sequential one (**NE1** and **NE2**). While we do not observe significant difference in the solution quality when using either default settings or extended running time (in both cases p-values of 0.4768 and 0.106 on random and structured instances, respectively), after tuning the numerical parameters there is a clear difference in favour of the sequential exploration **NE2**.

To examine the full potential of SA algorithms, we use Level 3 of our analysis and design automatically a completely new SA algorithm, which may combine the available algorithm components in previously un-explored ways. This is achieved by tuning seven categorical parameters, each related to one of the main components and 89 additional numerical parameters for the various options. In Figure 3 we compare the results obtained by these new SA algorithms (indicated by All) with the other ten algorithms tuned over ten seconds; the results of the Friedman rank sum test is reported in Table 4. On both instance classes the automatically generated SA algorithms (the rightmost boxplots) outperform the tuned existing algorithms. On the random instances the improvement is more substantial, while on the structured instances, while still statistically significant, it is less strong. This is because the results obtained on the structured instances by Bin and BR1 in particular are already close to the best known solutions, and there is therefore less margin for improvement.

In the Supplementary Material we include the additional comparison with the SA algorithms automatically generated for anytime behaviour. On the random instances the convergence of All is very good, even for the largest instances. On the structured instances of the main test set, instead, the convergence behaviour is very good, though not as strong as for the SAs for anytime behaviour, while on the larger instances the results are mixed: in two cases the convergence is much slower, in the third case instead the algorithm designed for solution quality converged to significantly better solutions than the SA designed for anytime behaviour, continuing discovering better solutions during the whole runtime.

6. SA algorithms for the PFSP

We have identified three main articles employing SA for the PFSP, which do not use objective-specific tricks or hybridizations. The oldest method by Osman and Potts (OP) uses problem-specific components [46], in particular the initial temperature **IT9**. For it we consider four variants, with random and sequential exploration (**NE1** and NE2, indicated by R and S in the algorithm name) and either insert or exchange neighbourhood (indicated by I or E in the algorithm name). The second algorithm, OS, is proposed by Ogbu and Smith [58]; in the original paper the authors also evaluate both the insert and the exchange neighbourhoods. Finally, in the more recent work CH [56] two variants use two different values for one parameter. The details of these algorithms are summarized in Table 5. In what follows, we discuss where appropriate separately the results for the instances whose size is smaller than all the instances of our training set (Tai001 to Tai030), the instances whose size is covered by the training set (Tai031 to Tai110), and the instances whose size is instead larger (Tai111 to Tai120). Separate plots for the subsets of instances are provided in the Supplementary Material.

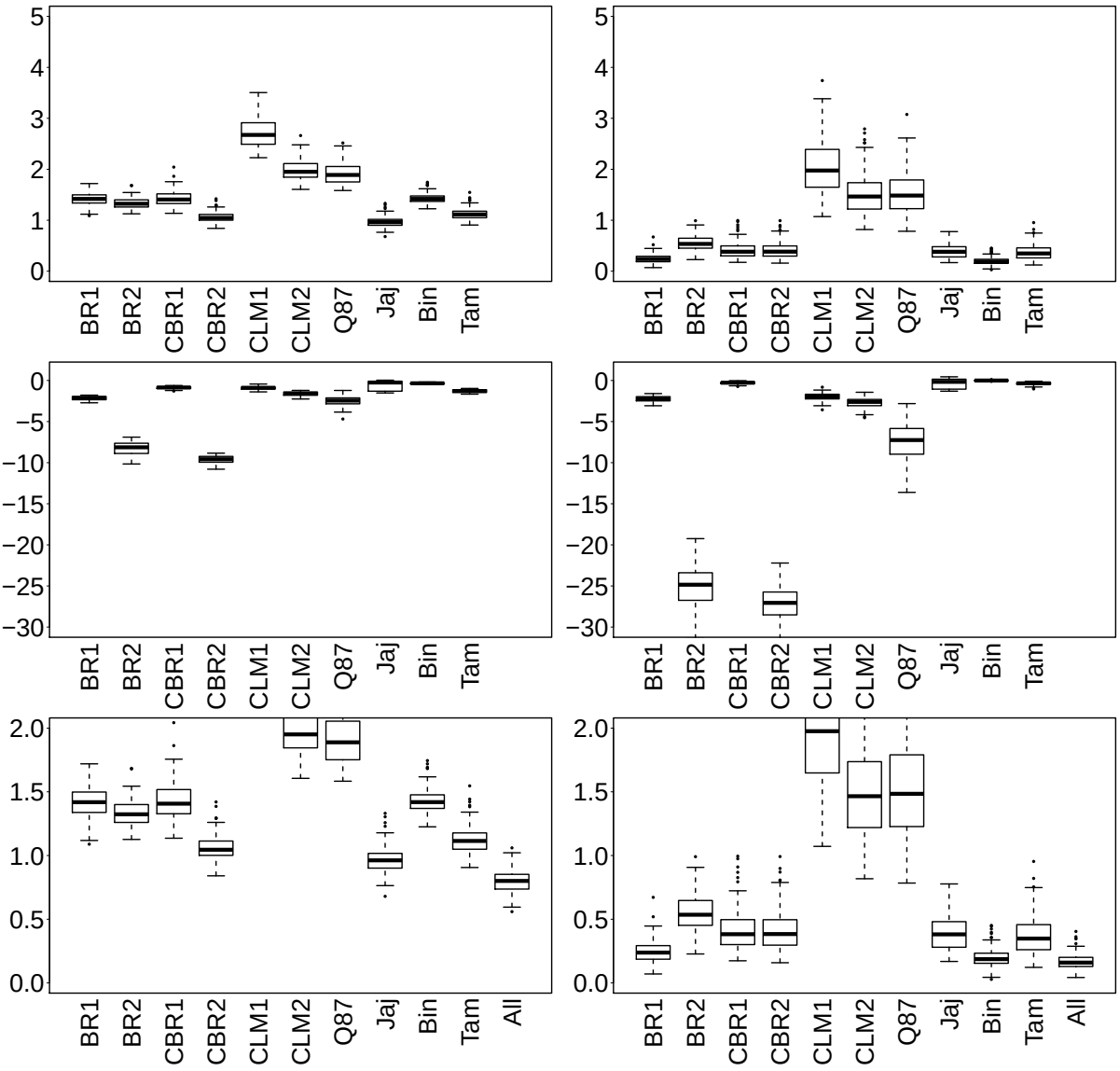

Figure 3: Average Relative Percentage Deviation (ARPD) from the best known solutions obtained by the algorithms with ten seconds of execution time on random (first row, left plot) and structured instances (right plot) after tuning their numerical parameters. In the second row we show the difference with respect to the results of Figure 2; a boxplot below 0 means that the performance improve. In the third row we include the results obtained by the algorithm generated when tuning the whole framework (`All`)

.

Algorithm	Temperature components	Exploration	Neighbourhood
`OP-IR`	**IT9**($k = 1$, $df = 5$) **AC1** **CS5**(a,b based on the termination) **TL1**($k = 1$) **TR1** **SC3**($n = 1$)	**NE1**	Exchange
`OP-SR`	**IT9**($k = 1$, $df = 5$) **AC1** **CS5**(a,b based on the termination) **TL1**($k = 1$) **TR1** **SC3**($n = 1$)	**NE1**	Insert
`OP-IO`	**IT9**($k = 1$, $df = 5$) **AC1** **CS5**(a,b based on the termination) **TL1**($k = 1$) **TR1** **SC3**($n = 1$)	**NE2**	Exchange
`OP-SO`	**IT9**($k = 1$, $df = 5$) **AC1** **CS5**(a,b based on the termination) **TL1**($k = 1$) **TR1** **SC3**($n = 1$)	**NE2**	Insert
`OS-E`	**IT6**($l = 10^4$, $k = 1$, $p = 0.2$) **AC5**($r = 0.75$) **CS2**($\alpha = 0.75$) **TL2**($k = 50$) **TR1** **SC4**($n = 15$)	**NE1**	Exchange
`OS-I`	**IT6**($l = 10^4$, $k = 1$, $p = 0.2$) **AC5**($r = 0.75$) **CS2**($\alpha = 0.75$) **TL2**($k = 50$) **TR1** **SC4**($n = 15$)	**NE1**	Insert
`CH1`	**IT1**($k = 1$) **AC3**($\phi = 001\%5$) **CS2**($\alpha = 0.999$) **TL1**($k = 1$) **TR1** **SC1**	**NE1**	Exchange
`CH5`	**IT1**($k = 1$) **AC3**($\phi = 005\%5$) **CS2**($\alpha = 0.999$) **TL1**($k = 1$) **TR1** **SC1**	**NE1**	Exchange

Table 5: Default settings for the eight algorithms of Section 6.

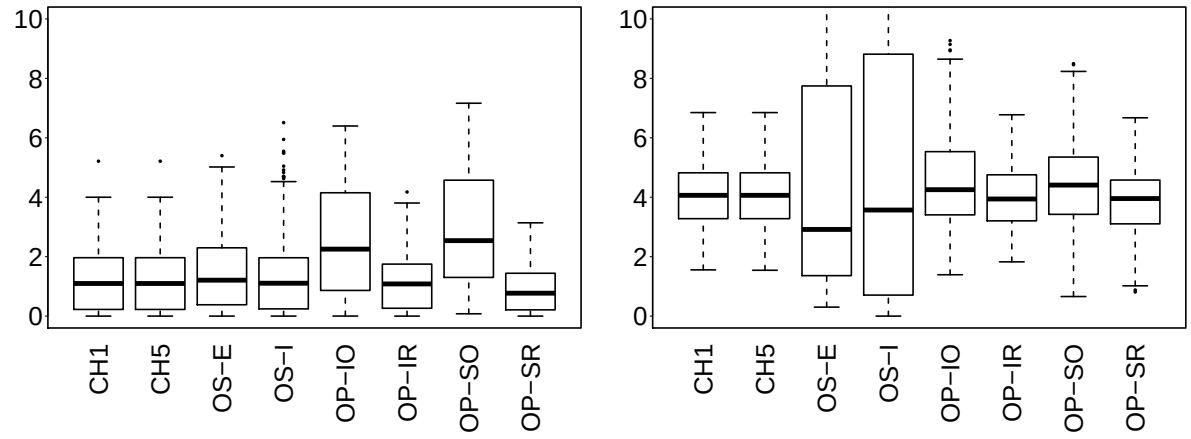

Figure 4: Average Relative Percentage Deviation (ARPD) from the best known solutions obtained by the algorithms on the whole Taillard benchmark under the MS and TCT objectives (left and right plot respectively).

In Figure 4 we report the results obtained by the algorithms in their original settings (Level 1 of the analysis), in terms of ARPD, while in Figure 5 we report the results obtained after the tuning (Level 2 of the analysis); the improvements of the tuning with respect to the default versions is reported in Figure 6. As `CH1` and `CH5` differ only for the value of one numerical parameter, they appear as a single algorithm `CH` in the tuning. In Figure 5 we also include the results obtained when automatically designing SA algorithms. Overall, we observe a more substantial improvement for the TCT, which is maybe explained by the usage of the MS objective in the original papers. In Table 6 we report the rankings obtained for the MS and TCT objectives by the algorithm default settings, and after the tuning.

The main observations for the various algorithms are the following. `CH1` and `CH5` both employ a bounded Metropolis condition and reach results that in the default settings are not significantly different. Probably this is due to a too low setting of the threshold for evaluating worsening solution. In fact, after the tuning of the `CH` algorithm, the tuned parameter settings promote exploration based on a higher initial temperature, a slow temperature decrease, and a much more relaxed threshold for considering worsening moves, improving especially the performance of `CH` on the small instances. Interestingly, while for the TCT objective the `CH` algorithms are the two worst ones, the tuned `CH` algorithm is the best among the already existing SA algorithms.

For the `OS` algorithms, the tuning improves strongly results on larger instances with 50 or more jobs, at the expense of the results on the small instances. A more detailed analysis of the tuned parameters confirms that except for some differences in the resulting temperature length, the main difference between `OS-I` and `OS-E` is the neighborhood used. Overall, the `OS` algorithms exhibit scaling issues even after the tuning, with the results on the larger instances not being of the same quality as on the other instances, especially for the TCT objective.

Of the four implementations from Osman and Potts, the two with a random neighborhood exploration perform better both, before and after the tuning. This is in striking contrast with the findings of the experiments on the QAP. Contrarily to the `OS` algorithms, the exchange neighbourhood outperforms the insert one for both objectives and both before and after the tuning.

As a next step, we apply Level 3 of our analysis, that is, we automatically design SA algorithms from the whole framework. The results are given in Figure 5 and Table 6 within the Tuned settings. On the MS objective the results are not statistically significantly different from the ones obtained

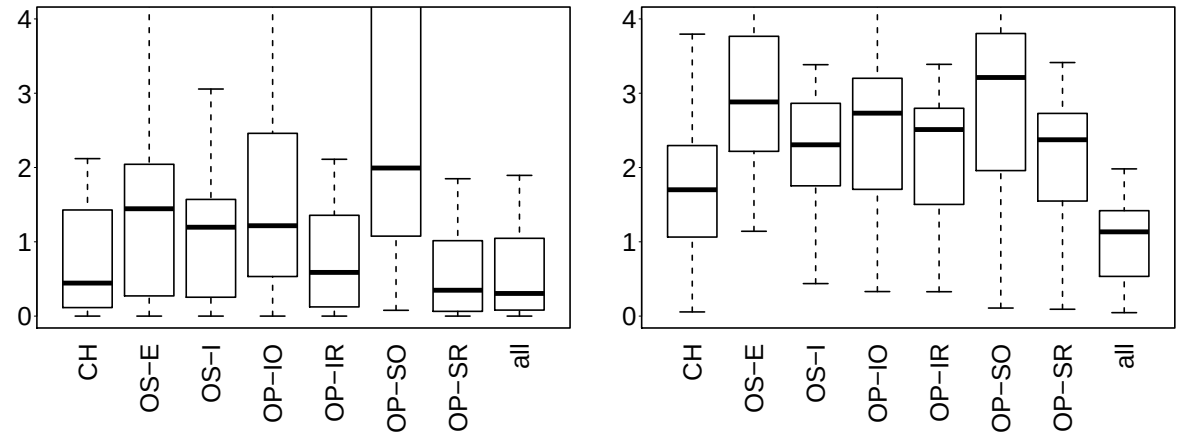

Figure 5: Average Relative Percentage Deviation (ARPD) from the best known solutions obtained by the algorithms after the tuning on the whole Taillard benchmark under the MS and TCT objectives (left and right plot respectively).

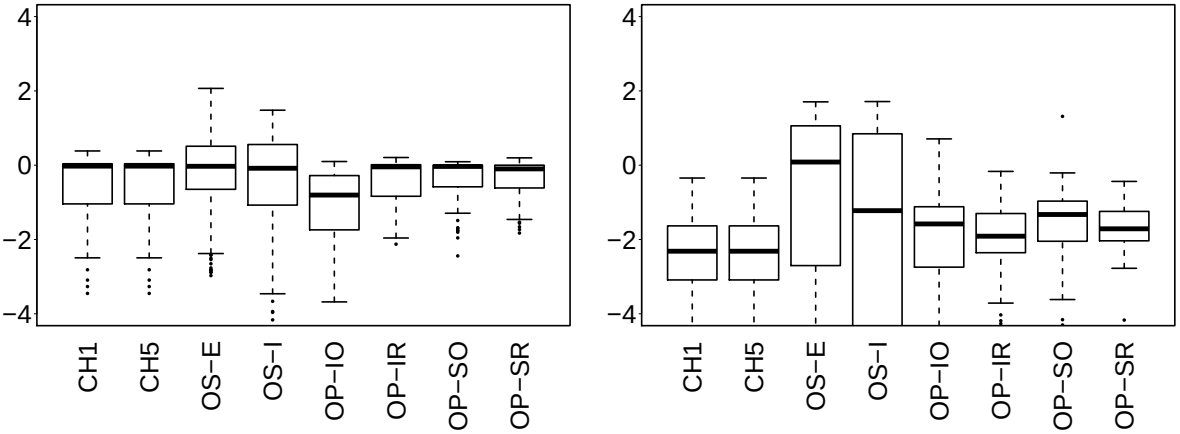

Figure 6: Improvement in terms of ARPD of the algorithms after the tuning on the whole Taillard benchmark under the MS and TCT objectives (left and right plot respectively).

by `OP-SR`. This may be due to the fact that under the given experimental conditions, the algorithms were already close to the best possible results obtainable by a simple SA with the insert or exchange neighbourhoods. On the TCT objective, instead, the automatically designed algorithms clearly outperform existing algorithms. In particular, we observe a much improved scaling behaviour, also on the larger instances.

On both objectives the anytime behaviour of the SA algorithms `All` is again good, sometimes even slightly better than the SAs designed for anytime behaviour (which, in turn, have a more regular convergence), due to the more complex tuning task required for the anytime behaviour. On the larges instances the search takes more time to discover a good solution, but from that point the behaviour is similar to the SAs tuned for anytime behaviour. The relative plots are given in the Supplementary Material.

Table 6: Comparison of the results of the Friedman rank sum test for the algorithms in their default settings and after the tuning of the numerical parameters (including the automatically generated SAs) on the Taillard benchmark instances for the PFSP-MS (top) and for the PFSP-TCT (bottom). Algorithms are ranked according to their results. Δ_R is the minimum rank-sum difference that indicates significant difference from the best one. Algorithms in boldface are significantly better than the following ones.

Settings	Δ_R	Algorithm ranking
Default	53.76	**OP-SR** (0), OP-IR (155.5), CH5 (179.5), CH1 (189.5), OS-I (243.5), OS-E (258), OP-IO (478.5), OP-SO (627.5)
Tuned	40.26	**OP-SR** (0), **all** (33.5), OP-IR (213), CH (233.5), OS-I (268), OS-E (476), OP-IO (516), OP-SO (700)
Default	72.47	**OS-E** (0), **OP-SR** (13), **OS-I** (71), OP-IR (122), OP-SO (134), OP-IO (156), CH1 (159), CH5 (161)
Tuned	46.82	**all** (0), CH (173), OP-SR (321), OP-IR (378), OS-I (444), OP-IO (547), OP-SO (612), OS-E (685)

7. Analysis of SA algorithms

We now analyze the data generated for configuring the `All` variants in Sections 5 and 6, trying to understand which algorithm components and features are important to make an SA algorithm high-performing. The first step is to observe which components and numerical parameters have the highest importance in the design process. We do so by training a random forest model, following the process outlined in Section 4, with which we get as a byproduct an estimate of the variable importance. Random forests [78] combine several decision trees, each built using a bootstrap sample of the training data. For each tree t the out-of-bag data (observations missing from the relative bootstrap sample, OOB_t) can be used as validation set, for which we measure (i) the error in the prediction, $errOOB_t$, and (ii) the error in the prediction, $\widetilde{errOOB}_t$, when the OOB_t data is perturbed (by permuting the values of each variable). The importance of a variable V_j is defined as the difference in the prediction error when the values for V_j are perturbed, averaged over the trees. For further insights about the importance of variables in random forest models we refer to [101]. The results of our analysis, normalized to obtain a probability, are reported in Table 7 for the four scenarios.

While some differences exist in the different scenarios, overall the single most important component is the acceptance criterion, along with some related numerical parameters, such as the LAHC tenure κ and the deviation factor for the RTR criterion. Another important parameter is the neighbourhood exploration, that is, how the algorithm chooses the next solution to evaluate; to this adds also the importance of the numerical parameter k for the neighborhood exploration options **NE3** or **NE4**. The impact of the acceptance is intuitive, as this component controls the exploration/exploitation tradeoff by determining which worsening moves are accepted. More surprising is maybe the high importance for the exploration, a component often neglected as it is common in the SA literature to assume a random selection in the neighbourhood. It is interesting to observe that the theoretically and experimentally most studied component of SA, the cooling scheme, is indeed important (as it influences the behaviour of the acceptance criterion), but not as important as commonly expected as in none of the scenarios it or its associated variables are among the four most highly ranked components or numerical parameters.

On the random QAP scenario the instance is the second most important factor, with a contribution more than four times higher than on the structured QAP scenario; the temperature restart

QAP Random		**QAP Structured**	
Acceptance Criterion	19.8%	Acceptance Criterion	36.64%
Instance	15.71%	**NE4** k	12%
NE4 k	8.43%	Exploration Criterion	9.33%
Exploration Criterion	8.17%	LAHC κ	5.19%
NE3 k	6.7%	RTR γ	3.97%
CS2 α	4.4%	Instance	3.72%
Cooling Scheme	4.02%	**CS2** α	3.6%
Temperature Restart	3.99%	**NE3** k	3.13%
IT5 k	3.53%	Initial Temperature	2.84%
RTR γ	3.2%	Cooling Scheme	2.72%
PFSP-MS		**PFSP-TCT**	
Exploration Criterion	20.65%	LAHC κ	16.02%
Instance	14.55%	Acceptance Criterion	15.92%
CS6 b	9.24%	Instance	15.33%
CS6 a	5.18%	Exploration Criterion	9.51%
Cooling Scheme	4.76%	**CS2** α	5.62%
CS2 α	4.47%	**NE4** k	3.95%
Acceptance Criterion	4.25%	**NE3** k	2.76%
CS5 b	2.65%	Temperature Length	2.43%
LAHC κ	2.46%	**AC5** r	2.37%
TL9 k	2.25%	Cooling Scheme	2.01%

Table 7: The ten most important components and numerical parameters on the four scenarios.

component also appears in the list. This is probably related to a more rugged landscape, where the lack of structure makes apparently the algorithms more sensitive to the numerical values of the algorithms. Also for the two PFSP scenarios, the instance factor has a high importance, which we here conjecture to be due to the rather different instance sizes in the training sets and the differences in the ratio jobs/machines.

Analogous tests on the data collected during the tuning for the anytime behaviour (for detailed results, see the supplementary material) show a higher importance of the instance factor. In particular, the more diverse the dataset, the more important is the instance factor (e.g. for the PFSP case). Another explanation for the stronger impact of the instance is that maximization of the hypervolume, which is used to measure anytime performance, is more sensitive to parameter settings in the sense that it reduces the set of configurations that result in high performance across various instances sizes or characteristics; hence it is more difficult for the configurator to reach configurations that generalize to the whole training set. Other components and parameters deemed important are mostly related to temperature initialization and update, which are relatively more important than in the design for solution quality scenarios.

A detailed analysis of the composition of the automatically generated algorithms is reported in the supplementary material. Here we highlight the most relevant results. The Metropolis criterion **AC1** is the most common choice in three out of four scenarios, appearing 8 out of 15 times on the structured QAP instances, 12 times on the random QAP ones (including the Bounded version twice), and 13 times on the PFSP-MS. On the PFSP-TCT, instead, LAHC is chosen 12 times, with the κ parameter ranging from 99 to 185 (with one outlier case being 5616). The other three algorithms for the PFSP-TCT use a Bounded Metropolis criterion, set to immediately discard

solutions whose cost is 0.53 to 0.68% higher than the incumbent; these three algorithms are all paired with a Geometric cooling scheme. The LAHC is chosen also in 4 cases in the structured QAP scenario, while 3 times Threshold Acceptance is chosen.

The sequential exploration criterion **NE2** is always chosen for the structured QAP instances; it is also chosen 8 times on the random QAP ones, with the **NE3** criterion selected the other 7 times. On both PFSP scenarios, instead, the random exploration **NE1** is chosen in all cases.

For the cooling scheme, the geometric one **CS2** is the only one chosen on the structured QAP scenario, and the one selected most often for the random QAP instances (7 times out of 15, along with five other cooling schemes) On the PFSP-MS the *Q8-7* and its original version Lundy-Mees are chosen respectively 8 and 3 times. For temperature length and restart, instead, we observe a wide range of choices, probably because the sequence of temperature values that yields a very good exploration/exploitation tradeoff can be given by different combinations of the components devoted to its update.

The outcome of the tuning for solution quality of the existing algorithms on the various scenarios is generally a strict acceptance of worsening moves, enforced by low values of the initial temperature and faster cooling than in the original settings. In other words, a well-performing SA algorithm quickly converges to good solutions, and then continues improving. This also explains the good performance of a simple scheme such as LAHC, that can never accept a solution worse than anything else encountered during the search, thus having limited though non-negligible exploration capabilities. The importance of a strong intensification is also reflected in the convergence profiles of the automatically generated algorithms (reported in the Supplementary Material), that are rather similar to the profiles of the SA algorithms automatically generated for anytime behaviour. To reach a better anytime behaviour and allow the algorithms to scale more regularly to larger instances, stricter acceptance criteria need to be set, e.g. even lower initial temperatures, a much shorter tenure of the LAHC acceptance criterion [44, 61], or a tight bound for the Bounded Metropolis condition.

Considering the configurations obtained from the various scenarios, we may conclude that there does not exist one single high-performance, "general" SA algorithm, but for different scenarios in part rather different settings turn out to be best, justifying the usage of an automatic algorithm configuration process. However, even for the same scenario, SA implementations may in part differ significantly as a good trade-off between search exploration and exploitation may be reached by different algorithm settings. This is particularly true when considering scenarios that only focus on the final solution quality. However, this is less the case when high anytime performance is required, apparently due to the more refined evaluation of the performance of a configuration.

Finally, let us mention also the discrepancy between our experimental results and the focus in many theoretical works about SA. As previously mentioned, SA theory focuses often on the aspects of the cooling process and gives conditions on the process to guarantee convergence of the algorithm to optimal solutions–typically achieved by very slow cooling schemes. This has probably contributed to a blind interpretation of the annealing metaphor, where a system is initially at a *high* temperature that *slowly* decreases over time. Consequently, many SA algorithms in the literature follow this folklore, despite a strong convergence and a not-necessarily random neighbourhood exploration have been already discussed in literature as beneficial (see e.g. [50] for the former, and [102, 103, 104] for the latter).

8. Conclusions

In this paper, we propose to exploit the available knowledge on algorithmic components and parameter setting strategies for metaheuristics in the form of automatically configurable frame-

works. As we have argued before, there are some inherent advantages associated to this such as (i) making these components and parameters explicitly available for further use, (ii) building a kind of collective memory of available algorithm options hardly any researcher alone may have readily available, (iii) allowing large-scale experimental analyses and the generation of knowledge under which circumstances which components will be most successful, and (iv) exploiting directly the recent advances in the automatic configuration of algorithms allowing to build potentially higher-performing algorithms than possible by pure manual intervention.

We have made our proposal concrete by building such a framework for Simulated Annealing (SA), one of the most widely studied metaheuristics. We have described the template from which we instantiate SA algorithms and detailed the set of algorithmic choices that are available in our current framework. Interestingly, also a number of methods that fit the general outline of an SA algorithm can be instantiated from the same template, including methods such as Threshold Accepting [42], Great Deluge [43], Record-To-Record Travel [43], or Late Acceptance Hill-Climbing [44, 61]. We have also experimentally studied various well-known SA variants for the quadratic assignment problem and two flow-shop scheduling problem variants both using default settings of the literature and different levels of automatic SA algorithm configuration. As maybe expected, the possibility of automatic configuration has shown to be very advantageous, allowing to derive SA algorithms that outperform the variants proposed in the literature even if the numerical parameters of these are also fine-tuned. The experimental data we obtained from the automatic configuration process have also shown to be useful to get insights into what makes a good SA algorithm. An importance analysis has identified the acceptance criterion and the neighbourhood exploration as the key important components of SA. Differently, when configuring for best reachable solution quality, the cooling scheme seemed relevant as it influences the behaviour of the acceptance criterion, but to a less degree than one would commonly expect.

Our work can be extended in a number of directions. A first one is to extend the experimental part to automatically generate SA algorithms for other problems and learn about possibly consistent choices and which algorithmic components are the most relevant ones; the knowledge obtained may in turn provide prior biases for the configuration process and lead to the development of new alternative algorithmic components. A second one is to extend our approach to other metaheuristics and to generate extensive such frameworks. Ideally, these extensions would be generated within a same framework so that possibly rich hybrids among these methods may be generated, as suggested in [105, 106].

Acknowledgments

This research and its results have been made possible through funding from the COMEX project (P7/36) within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a Research Director. The authors thank Federico Pagnozzi for helping and assistance in implementing our solutions using the EMILI framework.

[1] H. H. Hoos, T. Stützle, Stochastic Local Search—Foundations and Applications, Morgan Kaufmann Publishers, San Francisco, CA, 2005.

[2] M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, 2nd Edition, Vol. 146 of International Series in Operations Research & Management Science, Springer, New York, NY, 2010.

[3] D. B. Fogel, A. J. Owens, M. J. Walsh, Artificial Intelligence Through Simulated Evolution, John Wiley & Sons, 1966.

[4] I. Rechenberg, Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Frommann-Holzboog, Stuttgart, Germany, 1973.

[5] H.-P. Schwefel, Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie, Birkhäuser, Basel, Switzerland, 1977.

[6] J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.

[7] F. Glover, Heuristics for integer programming using surrogate constraints, Decision Sciences 8 (1977) 156–166.

[8] F. Glover, Future paths for integer programming and links to artificial intelligence, Computers & Operations Research 13 (5) (1986) 533–549.

[9] P. Hansen, B. Jaumard, Algorithms for the maximum satisfiability problem, Computing 44 (1990) 279–303.

[10] K. Sörensen, M. Sevaux, F. Glover, A history of metaheuristics, in: R. Martí, P. M. Pardalos, M. G. C. Resende (Eds.), Handbook of Heuristics, Springer International Publishing, 2018, pp. 1–27, to appear.

[11] K. Sörensen, Metaheuristics—the metaphor exposed, International Transactions in Operational Research 22 (1) (2015) 3–18. doi:10.1111/itor.12001.

[12] D. Weyland, A rigorous analysis of the harmony search algorithm: How the research community can be misled by a "novel" methodology, International Journal of Applied Metaheuristic Computing 12 (2) (2010) 50–60.

[13] D. Weyland, A critical analysis of the harmony search algorithm: How not to solve Sudoku, Operations Research Perspectives 2 (2015) 97–105.

[14] A. G. Nikolaev, S. H. Jacobson, Simulated annealing, in: Gendreau and Potvin [2], pp. 1–39.

[15] E. H. L. Aarts, J. H. M. Korst, W. Michiels, Simulated annealing, in: Search Methodologies, Springer, 2005, pp. 187–210.

[16] H. H. Hoos, Programming by optimization, Communications of the ACM 55 (2) (2012) 70–80. doi:10.1145/2076450.2076469.

[17] F. Hutter, H. H. Hoos, K. Leyton-Brown, T. Stützle, ParamILS: an automatic algorithm configuration framework, Journal of Artificial Intelligence Research 36 (2009) 267–306.

[18] F. Hutter, H. H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: C. A. Coello Coello (Ed.), Learning and Intelligent Optimization, 5th International Conference, LION 5, Vol. 6683 of Lecture Notes in Computer Science, Springer, Heidelberg, Germany, 2011, pp. 507–523.

[19] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, M. Birattari, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58. doi:10.1016/j.orp.2016.09.002.

[20] A. R. KhudaBukhsh, L. Xu, H. H. Hoos, K. Leyton-Brown, SATenstein: Automatically building local search SAT solvers from components, in: C. Boutilier (Ed.), Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09), AAAI Press, Menlo Park, CA, 2009, pp. 517–524.

[21] A. R. KhudaBukhsh, L. Xu, H. H. Hoos, K. Leyton-Brown, SATenstein: Automatically building local search SAT Solvers from Components, Artificial Intelligence 232 (2016) 20–42.

[22] M. López-Ibáñez, T. Stützle, The automatic design of multi-objective ant colony optimization algorithms, IEEE Transactions on Evolutionary Computation 16 (6) (2012) 861–875. doi:10.1109/TEVC.2011.2182651.

[23] T. Liao, T. Stützle, M. A. Montes de Oca, M. Dorigo, A unified ant colony optimization algorithm for continuous optimization, European Journal of Operational Research 234 (3) (2014) 597–609.

[24] L. C. T. Bezerra, M. López-Ibáñez, T. Stützle, Automatic component-wise design of multi-objective evolutionary algorithms, IEEE Transactions on Evolutionary Computation 20 (3) (2016) 403–417. doi:10.1109/TEVC.2015.2474158.

[25] A. Franzin, T. Stützle, Revisiting simulated annealing: a component-based analysis: Supplementaty material, http://iridia.ulb.ac.be/supp/IridiaSupp2018-001 (2018).

[26] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. Teller, E. Teller, Equation of state calculations by fast computing machines, Journal of Chemical Physics 21 (1953) 1087–1092.

[27] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[28] V. Černý, A thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm, Journal of Optimization Theory and Applications 45 (1) (1985) 41–51.

[29] M. Lundy, A. Mees, Convergence of an annealing algorithm, Mathematical Programming 34 (1) (1986) 111–124.

[30] B. Hajek, Cooling schedules for optimal annealing, Mathematics of Operations Research 13 (2) (1988) 311–329.

[31] D. Mitra, F. Romeo, A. Sangiovanni-Vincentelli, Convergence and finite-time behavior of simulated annealing, in: Decision and Control, 1985 24th IEEE Conference on, IEEE, 1985, pp. 761–767.

[32] F. Romeo, A. Sangiovanni-Vincentelli, A theoretical framework for simulated annealing, Algorithmica 6 (1-6) (1991) 302–345.

[33] D. Henderson, S. H. Jacobson, A. W. Johnson, The theory and practice of simulated annealing, in: Handbook of Metaheuristics, Springer, 2003, pp. 287–319.

[34] P. J. M. van Laarhoven, E. H. L. Aarts, Simulated Annealing: Theory and Applications, Vol. 37, Springer, 1987.

[35] R. W. Eglese, Simulated annealing: a tool for operational research, European Journal of Operational Research 46 (3) (1990) 271–281.

[36] M. Jerrum, Large cliques elude the metropolis process, Random Structures & Algorithms 3 (4) (1992) 347–359.

[37] A. W. Johnson, S. H. Jacobson, On the convergence of generalized hill climbing algorithms, Discrete Applied Mathematics 119 (1) (2002) 37–57.

[38] J. E. Orosz, S. H. Jacobson, Analysis of static simulated annealing algorithms, Journal of Optimization Theory and Applications 115 (1) (2002) 165–182.

[39] H. Cohn, M. J. Fielding, Simulated annealing: Searching for an optimal temperature, SIAM Journal on Optimization 9 (3) (1999) 779–802.

[40] M. J. Fielding, Simulated annealing with an optimal fixed temperature, SIAM Journal on Optimization 11 (2) (2000) 289–307.

[41] P. Moscato, J. F. Fontanari, Stochastic versus deterministic update in simulated annealing, Physics Letters A 146 (4) (1990) 204–208.

[42] G. Dueck, T. Scheuer, Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing, Journal of Computational Physics 90 (1) (1990) 161–175.

[43] G. Dueck, New optimization heuristics: the great deluge algorithm and the record-to-record travel, Journal of Computational Physics 104 (1) (1993) 86–92.

[44] E. K. Burke, Y. Bykov, The late acceptance hill-climbing heuristic, Tech. Rep. CSM-192, University of Stirling (2012).

[45] C. Andrieu, N. de Freitas, A. Doucet, M. I. Jordan, An introduction to MCMC for machine learning, Machine Learning 50 (1-2) (2003) 5–43.

[46] I. H. Osman, C. N. Potts, Simulated annealing for permutation flow-shop scheduling, Omega 17 (6) (1989) 551–557.

[47] M. S. Hussin, T. Stützle, Tabu search vs. simulated annealing for solving large quadratic assignment instances, Computers & Operations Research 43 (2014) 286–291.

[48] R. E. Burkard, F. Rendl, A thermodynamically motivated simulation procedure for combinatorial optimization problems, European Journal of Operational Research 17 (2) (1984) 169–174. doi:10.1016/0377-2217(84)90231-5.

[49] D. T. Connolly, An improved annealing scheme for the qap, European Journal of Operational Research 46 (1) (1990) 93–100.

[50] D. S. Johnson, C. R. Aragon, L. A. McGeoch, C. Schevon, Optimization by simulated annealing: An experimental evaluation: Part I, graph partitioning, Operations Research 37 (6) (1989) 865–892.

[51] D. S. Johnson, C. R. Aragon, L. A. McGeoch, C. Schevon, Optimization by simulated annealing: An experimental evaluation: Part II, graph coloring and number partitioning, Operations Research 39 (3) (1991) 378–406.

[52] K. Y. Tam, A simulated annealing algorithm for allocating space to manufacturing cells, International Journal of Production Research 30 (1) (1992) 63–87.

[53] A. Misevičius, A modified simulated annealing algorithm for the quadratic assignment problem, Informatica 14 (4) (2003) 497–514.

[54] S. Jajodia, I. Minis, G. Harhalakis, J.-M. Proth, CLASS: computerized layout solutions using simulated annealing, International Journal of Production Research 30 (1) (1992) 95–108.

[55] H. Ishibuchi, S. Misaki, H. Tanaka, Modified simulated annealing algorithms for the flow shop sequencing problem, European Journal of Operational Research 81 (2) (1995) 388–398.

[56] R.-M. Chen, F.-R. Hsieh, An exchange local search heuristic based scheme for permutation flow shop problems, Applied Mathematics & Information Sciences 8 (1) (2014) 209–215.

[57] I. O. Bohachevsky, M. E. Johnson, M. L. Stein, Generalized simulated annealing for function optimization, Technometrics 28 (3) (1986) 209–217.

[58] F. A. Ogbu, D. K. Smith, The application of the simulated annealing algorithm to the solution of the n/m/C max flowshop problem, Computers & Operations Research 17 (3) (1990) 243–253.

[59] P. Moscato, On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, Caltech Concurrent Computation Program, C3P Report 826, Caltech (1989).

[60] B. Hajek, G. Sasaki, Simulated annealing–to cool or not, System & Control Letters 12 (5) (1989) 443–447.

[61] E. K. Burke, Y. Bykov, The late acceptance hill-climbing heuristic, European Journal of Operational Research 258 (1) (2017) 70–78.

[62] J. Appleby, D. Blake, E. Newman, Techniques for producing school timetables on a computer and their application to other scheduling problems, The Computer Journal 3 (4) (1961) 237–245.

[63] S. Geman, D. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, IEEE Transactions on Pattern Analysis and Machine Intelligence 6 (6) (1984) 721–741.

[64] P. N. Strenski, S. Kirkpatrick, Analysis of finite length annealing schedules, Algorithmica 6 (1-6) (1991) 346–366.

[65] H. Szu, R. Hartley, Fast simulated annealing, Physics Letters A 122 (3) (1987) 157–162.

[66] K. Andersen, R. V. V. Vidal, V. B. Iversen, Design of a teleprocessing communication network using simulated annealing, in: Vidal [107], pp. 201–215.

[67] S. Kirkpatrick, Optimization by simulated annealing: Quantitative studies, Journal of Statistical Physics 34 (5-6) (1984) 975–986.

[68] M. Jerrum, A. Sinclair, The Markov chain Monte Carlo method: an approach to approximate counting and integration, in: D. S. Hochbaum (Ed.), Approximation Algorithms For NP-hard Problems, PWS Publishing Co., 1996, pp. 482–520.

[69] T. C. Hu, A. B. Kahng, C.-W. A. Tsao, Old bachelor acceptance: A new class of non-monotone threshold accepting methods, ORSA Journal on Computing 7 (4) (1995) 417–425.

[70] D. Abramson, Constructing school timetables using simulated annealing: Sequential and parallel algorithms, Management Science 37 (1) (1991) 98–113.

[71] J. Rose, W. Klebsch, J. Wolf, Temperature measurement and equilibrium dynamics of simulated annealing placements, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 9 (3) (1990) 253–259.

[72] A. Souilah, Simulated annealing for manufacturing systems layout design, European Journal of Operational Research 82 (3) (1995) 592–614.

[73] D. Abramson, M. K. Amoorthy, H. Dang, Simulated annealing cooling schedules for the school timetabling problem, Asia-Pacific Journal of Operational Research 16 (1) (1999) 1–22.

[74] F. Pagnozzi, T. Stützle, Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems, Tech. Rep. TR/IRIDIA/2018-005, IRIDIA, Université Libre de Bruxelles, Belgium (Apr. 2018).
URL http://iridia.ulb.ac.be/IridiaTrSeries/link/IridiaTr2018-005.pdf

[75] T. Stützle, Some thoughts on engineering stochastic local search algorithms, in: A. Viana, et al. (Eds.), Proceedings of the EU/MEeting 2009: Debating the future: new areas of application and innovative approaches, 2009, pp. 47–52.

[76] S. Zilberstein, Using anytime algorithms in intelligent systems, AI Magazine 17 (3) (1996) 73–83.

[77] M. López-Ibáñez, T. Stützle, Automatically improving the anytime behaviour of optimisation algorithms, European Journal of Operational Research 235 (3) (2014) 569–582. doi:10.1016/j.ejor.2013.10.043.

[78] L. Breiman, Random forests, Machine Learning 45 (1) (2001) 5–32. doi:10.1023/A:1010933404324.

[79] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, M. Birattari, The irace package, iterated race for automatic algorithm configuration, Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011).
URL http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf

[80] P. Balaprakash, M. Birattari, T. Stützle, Improvement strategies for the F-race algorithm: Sampling design and iterative refinement, in: T. Bartz-Beielstein, M. J. Blesa, C. Blum, B. Naujoks, A. Roli, G. Rudolph, M. Sampels (Eds.), Hybrid Metaheuristics, Vol. 4771 of Lecture Notes in Computer Science, Springer, Heidelberg, Germany, 2007, pp. 108–122.

[81] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-race and iterated F-race: An overview, in: T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (Eds.), Experimental Methods for the Analysis of Optimization Algorithms, Springer, Berlin, Germany, 2010, pp. 311–336.

[82] M. N. Wright, A. Ziegler, ranger: A fast implementation of random forests for high dimensional data in C++ and R, Journal of Statistical Software 77 (1) (2017) 1–17. doi:10.18637/jss.v077.i01.

[83] T. C. Koopmans, M. J. Beckmann, Assignment problems and the location of economic activities, Econometrica 25 (1957) 53–76.

[84] R. E. Burkard, E. Çela, P. M. Pardalos, L. S. Pitsoulis, The quadratic assignment problem, in: P. M. Pardalos, D.-Z. Du (Eds.), Handbook of Combinatorial Optimization, Vol. 2, Kluwer Academic Publishers, 1998, pp. 241–338.

[85] M. Fischetti, M. Monaci, D. Salvagnin, Three ideas for the quadratic assignment problem, Operations Research 60 (4) (2012) 954–964.

[86] É. D. Taillard, Comparison of iterative searches for the quadratic assignment problem, Location Science 3 (2) (1995) 87–105.

[87] M. R. Garey, D. S. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling, Mathematics of Operations Research 1 (1976) 117–129.

[88] J. M. Framiñán, R. Leisten, R. Ruiz, Manufacturing Scheduling Systems: An Integrated View on Models, Methods, and Tools, Springer, New York, NY, 2014.

[89] M. L. Pinedo, Scheduling: Theory, Algorithms, and Systems, 4th Edition, Springer, New York, NY, 2012.

[90] V. Fernandez-Viagas, R. Ruiz, J. M. Framiñán, A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation, European Journal of Operational Research 257 (3) (2017) 707–721.

[91] Q.-K. Pan, R. Ruiz, A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime, Computers & Operations Research 40 (1) (2013) 117–128.

[92] Q.-K. Pan, R. Ruiz, Local search methods for the flowshop scheduling problem with flowtime minimization, European Journal of Operational Research 222 (1) (2012) 31–43.

[93] M. Nawaz, E. Enscore, Jr, I. Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, Omega 11 (1) (1983) 91–95.

[94] F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, From grammars to parameters: Automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness, in: P. M. Pardalos, G. Nicosia (Eds.), Learning and Intelligent Optimization, 7th International Conference, LION 7, Vol. 7997 of Lecture Notes in Computer Science, Springer, Heidelberg, Germany, 2013, pp. 321–334. doi:10.1007/978-3-642-44973-4_36.

[95] É. D. Taillard, Benchmarks for basic scheduling problems, European Journal of Operational Research 64 (2) (1993) 278–285.

[96] E. Vallada, R. Ruiz, J. M. Framiñán, New hard benchmark for flowshop scheduling problems minimising makespan, European Journal of Operational Research 240 (3) (2015) 666–677. doi:10.1016/j.ejor.2014.07.033.

[97] J. Paulli, A computational comparison of simulated annealing and tabu search applied to the quadratic assignment problem, in: Vidal [107], pp. 85–102.

[98] E. Çela, The Quadratic Assignment Problem: Theory and Algorithms, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[99] R. Battiti, G. Tecchiolli, Simulated annealing and tabu search in the long run: A comparison on QAP tasks, Computer and Mathematics with Applications 28 (6) (1994) 1–8. doi:10.1016/0898-1221(94)00147-2.

[100] G. Paul, Comparative performance of tabu search and simulated annealing heuristics for the quadratic assignment problem, Operations Research Letters 38 (6) (2010) 577–581.

[101] R. Genuer, J.-M. Poggi, C. Tuleau-Malot, Variable selection using random forests, Pattern Recognition Letters 31 (14) (2010) 2225–2236.

[102] B. L. Fox, Uniting probabilistic methods for optimization, in: Proceedings of the 24th conference on Winter simulation, ACM, 1992, pp. 500–505.

[103] B. L. Fox, Integrating and accelerating tabu search, simulated annealing, and genetic algorithms, Annals of Operations Research 41 (2) (1993) 47–67.

[104] B. L. Fox, Simulated annealing: folklore, facts, and directions, in: Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing, Springer, 1995, pp. 17–48.

[105] M. López-Ibáñez, M.-E. Kessaci, T. Stützle, Automatic design of hybrid metaheuristics from algorithmic components, Tech. Rep. TR/IRIDIA/2017-012, IRIDIA, Université Libre de Bruxelles, Belgium (Dec. 2017).
URL http://iridia.ulb.ac.be/IridiaTrSeries/link/IridiaTr2017-012.pdf

[106] M.-E. Marmion, F. Mascia, M. López-Ibáñez, T. Stützle, Automatic design of hybrid stochastic local search algorithms, in: M. J. Blesa, C. Blum, P. Festa, A. Roli, M. Sampels (Eds.), Hybrid Metaheuristics, Vol. 7919 of Lecture Notes in Computer Science, Springer, Heidelberg, Germany, 2013, pp. 144–158. doi:10.1007/978-3-642-38516-2_12.

[107] R. V. V. Vidal (Ed.), Applied Simulated Annealing, Springer, 1993.